Parson’s Problem Appliance for Learning Management Systems

Testing Document

Anthony Narlock,
Jaden Rodriguez,
Shen Lua,
Stephanie Ye

PPALMS Testing Document v0.0.1
November 28th, 2022

Revision History

Version	Reason for Changes	Date
0.0.1	Initial Commit	November 28th, 2022

Contents:

1. Purpose

Extensive unit testing of the Parson's Problem Appliance for Learning Management Systems (PPALMS) application is integral to ensure that the system's requirements are met and flaws of the system are identified before deployment. The Testing Document serves as a demonstration of the functions and capabilities of the PPALMS application as stated in the requirements and design documents. Testing is performed in a manner such that probable scenarios and edge cases are considered in the process to ensure system robustness in an event of uncertain user input.

1.1 Background Knowledge

Individuals who have read the preceding requirements and design document, would have a general expectation of the PPALMS system application's functions and capabilities. In an effort to maintain relatability, the tests utilize a naming convention that is identical to how the tests were written using the JUnit testing framework. In this document, the tests are organized by class name as defined in the design document. Specifically, both the requirements and design documents were of version 0.0.2.

1.2 Testing Conventions

The testing documentation is structured to improve readability by mimicking the structure of the requirements and design document. The tests incorporated are a mixture of both *functional* and *standard unit testing*. Functional, for the purpose of demonstrating functionality of the PPALMS system application from an end-user's perspective by testing the graphical user interface directly, and unit tests written with the Java JUnit testing framework to ensure each individual component of the system functions as expected. It is important to note that since the functional tests are not provided in the JUnit tests for the application, the naming conventions are different. Functional tests are written in this document in regular English and the unit tests are written in this document with respect to their corresponding test method. Additionally, since functional tests are not done in an automated fashion like standard unit testing, we provide the result of executing them inside of test definition. The testing document concludes with an extensive end-to-end test scenario, which is inclusive of the setup, step-by-step execution procedure of the PPALMS system application and necessary verification checks. These are traced back to the requirements of the PPALMS system, for which are exercised by the scenario.

The tests are written and documented in accordance with the respective application code segments by the developer and tester. Therefore, the "owner" of each test will belong to the respective development engineer and testing engineer. Under the condition of one owner, this means that the developer and unit tester were the same engineer. Furthermore, the functional tests were performed manually by the respective owners utilizing the application, while the standard unit tests utilized the JUnit testing framework. Additionally, the names of the tests reflect the Java coding conventions described in the design document.

2. Unit Test Cases

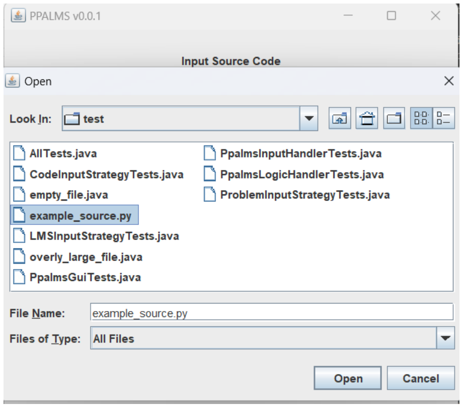
In this section, the JUnit testing framework was utilized for the set of unit tests. As stated in the previous section, the use of functional tests were also utilized for parts of the code that required sufficient manual testing. The tests that utilize manual testing are indicated as such.

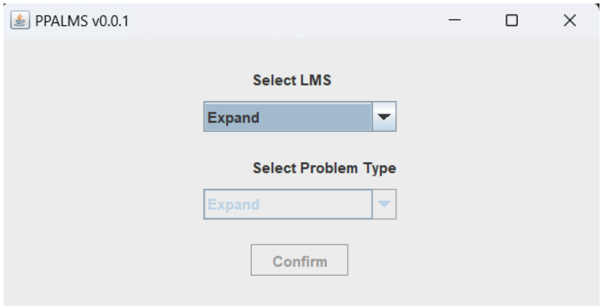
The purpose of creating unit tests is to isolate code and test its functionality to ensure that it works as intended. The unit tests sections will be divided by class of the system that has been tested. The following unit tests will follow the testing conventions described in the example documentation test case. A subsequent execution report of the tests will be provided in section 3.

2.1 PpalmsGui Tests

2.1.1 Functional Test Cases

<u>Functional Test Case #1 Successful File Input</u>
Purpose: To test successfully uploading a valid SourceCode file.
Owners: Anthony Narlock, Jaden Rodriguez, Shen Lua, Stephanie Ye
Expected Results: No error dialog is displayed in the PPALMS application and view strategy changes from CodeInputStrategy to LMSInputStrategy.
Test Data: The uploaded valid source code file.
Dependencies: A PPALMS JAR file is present and able to launch on the user's system.

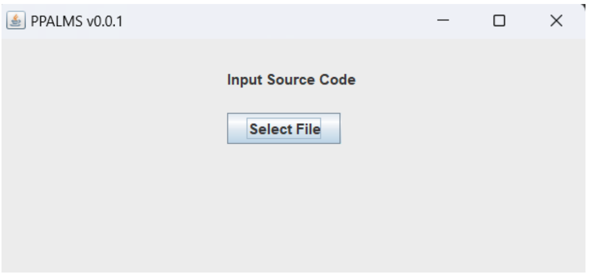
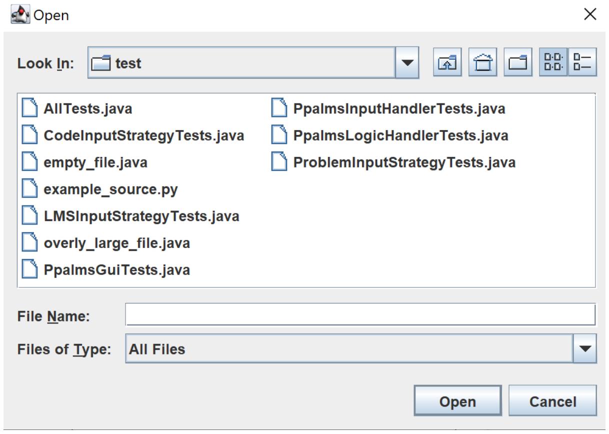
testUpdateViewStrategy	PPALMS v0.0.1
Step 1: Click on the "Select File" button.	
Expected Result	A filesystem window will be opened as an overlay.
Execution Report	Window opens as expected.  PPALMS v0.0.1 Input Source Code Open Look In: test AllTests.java CodeInputStrategyTests.java empty_file.java example_source.py LMSInputStrategyTests.java overly_large_file.java PpalmsGuiTests.java PpalmsInputHandlerTests.java PpalmsLogicHandlerTests.java ProblemInputStrategyTests.java File Name: example_source.py Files of Type: All Files Open Cancel

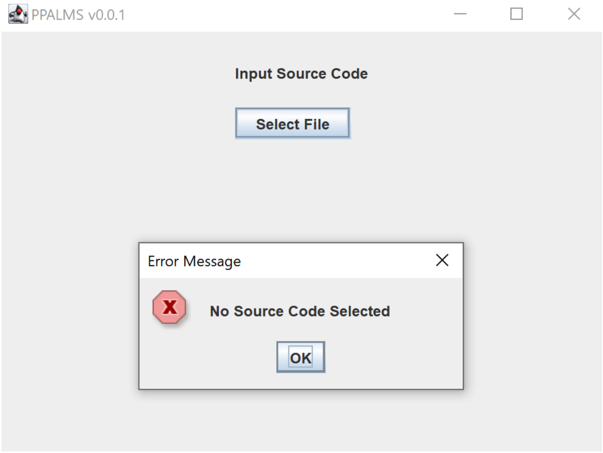
Step 2: Upload a valid SourceCode file.	
Expected Result	No error dialog is displayed in the PPALMS application and the view strategy changes from CodeInputStrategy to LMSInputStrategy.
Execution Report	View changes as expected without error dialog.  PPALMS v0.0.1 Select LMS Expand Select Problem Type Expand Confirm

<u>Functional Test Case #2 Unsuccessful File Input - No Source Code Selected</u>

Purpose: To test how the GUI responds to an unsuccessful file input where no source code is selected. That is, when the user exits the file chooser without specifying a file to open.
Owners: Anthony Narlock, Jaden Rodriguez, Shen Lua, Stephanie Ye
Expected Results: An error message is displayed indicating “No Source Code Selected”.
Test Data: N/A
Dependencies: The PPALMS application has been launched by the user and the CodeInputStrategy is displayed.

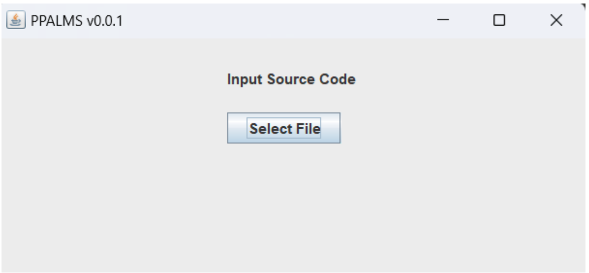
No Source Code Selected	PPALMS v0.0.1
Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy.
Execution Report	Application window opens to code input as expected

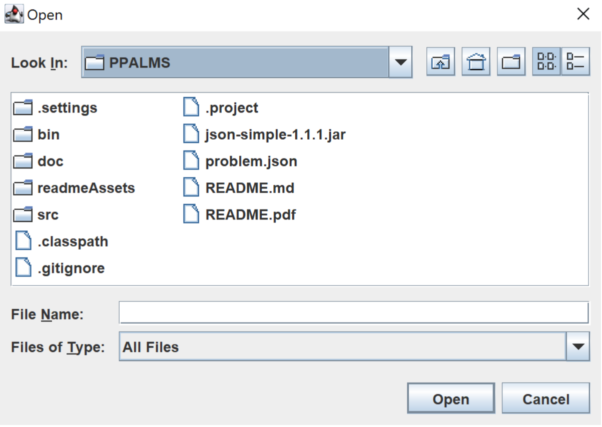
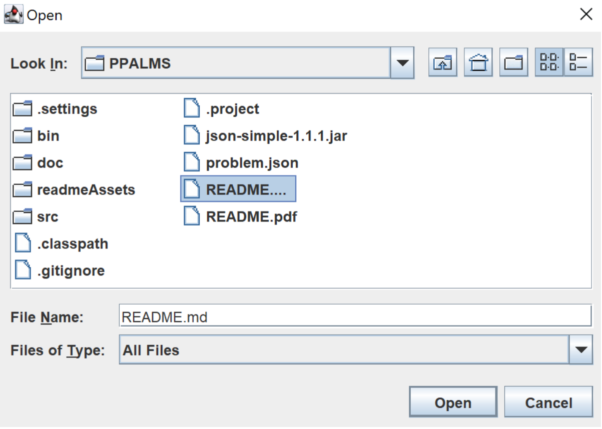
	
Step 2: Click the Select File button	
Expected Result	The PPALMS file system window has opened and the user is able to view the CodeInputStrategy.
Execution Report	The file system window opens as expected.  Look In: test AllTests.java CodeInputStrategyTests.java empty_file.java example_source.py LMSInputStrategyTests.java overly_large_file.java PpalmsGuiTests.java PpalmsInputHandlerTests.java PpalmsLogicHandlerTests.java ProblemInputStrategyTests.java File Name: Files of Type: All Files
Step 3: Click the cancel button on the file system menu.	
Expected Result	No Source Code Selected Error Message will appear and "OK" can be clicked to restart the Input Source Code operation.
Execution Report	Error message with correct prompt and "OK" option appear. Source code input restarts upon clicking.

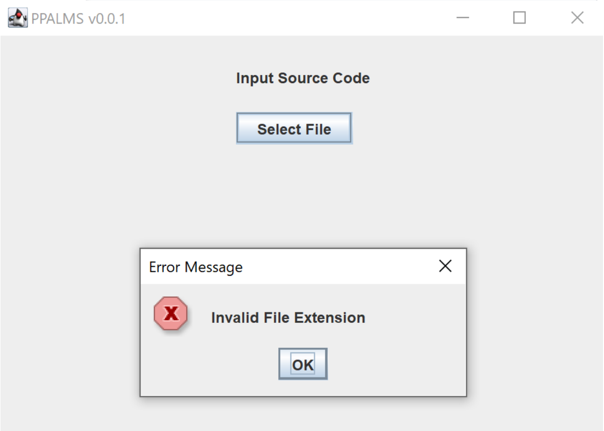
	 PPALMS v0.0.1 Input Source Code Select File Error Message OK No Source Code Selected

Functional Test Case #3 Unsuccessful File Input - Invalid File Extension Selected

Purpose: This functional tests manually how the GUI responds to an unsuccessful file input where a source code file with an invalid file extension is selected.
Owners: Anthony Narlock, Jaden Rodriguez, Shen Lua, Stephanie Ye
Expected Results: An error message is displayed indicating "Invalid File Extension".
Test Data: File with invalid extension (used READ.md).
Dependencies: The PPALMS application has just been launched by the user and the CodeInputStrategy is displayed.

Invalid File Extension Selected	PPALMS v0.0.1
Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy.
Execution Report	Application window opens to code input as expected. 

Step 2: Click the Select File button	
Expected Result	The PPALMS file system window has opened and the user is able to view the CodeInputStrategy.
Execution Report	The file system window opens as expected.  Look In: PPALMS File Name: Files of Type: All Files Open .settings bin doc readmeAssets src .classpath .gitignore .project json-simple-1.1.1.jar problem.json README.md README.pdf Open Cancel
Step 3: Select an invalid file (in this case, used README.md)	
Expected Result	The selected source code file appears in the File Name text field and is highlighted
Execution Report	The source code file appears in File Name text field.  Look In: PPALMS File Name: README.md Files of Type: All Files Open .settings bin doc readmeAssets src .classpath .gitignore .project json-simple-1.1.1.jar problem.json README.... README.pdf Open Cancel
Step 4: Click the “Open” button	

Expected Result	Invalid File Extension Error Message will appear and “OK” can be clicked to restart the Input Source Code operation.
Execution Report	Invalid File Extension Error Message appears.  PPALMS v0.0.1 Input Source Code Select File Error Message OK Invalid File Extension

<u>Functional Test Case #4 Unsuccessful File Input - Empty Source Code File Selected</u>

Purpose: This functional tests manually how the GUI responds to an unsuccessful file input where a source code file is empty

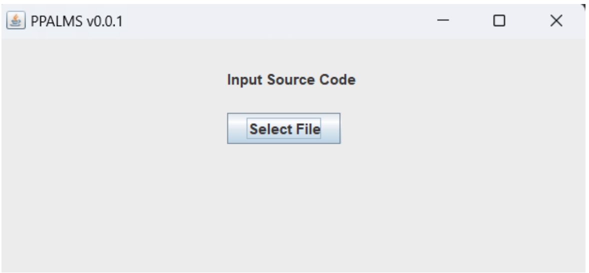
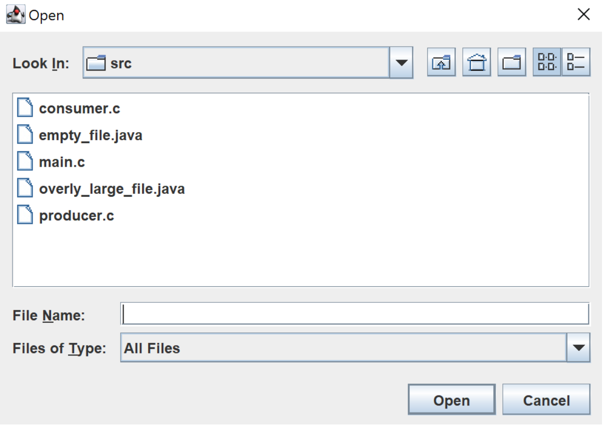
Owners: Anthony Narlock, Jaden Rodriguez, Shen Lua, Stephanie Ye

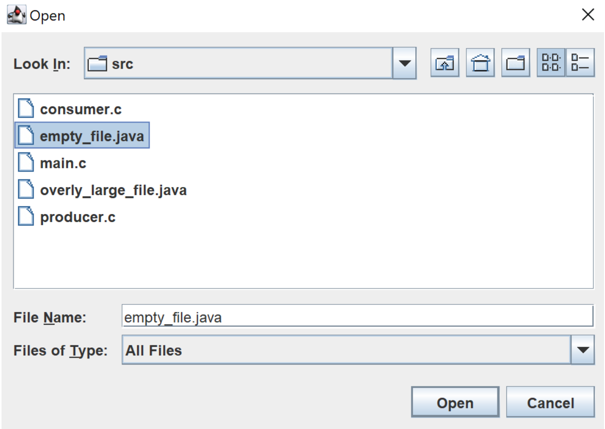
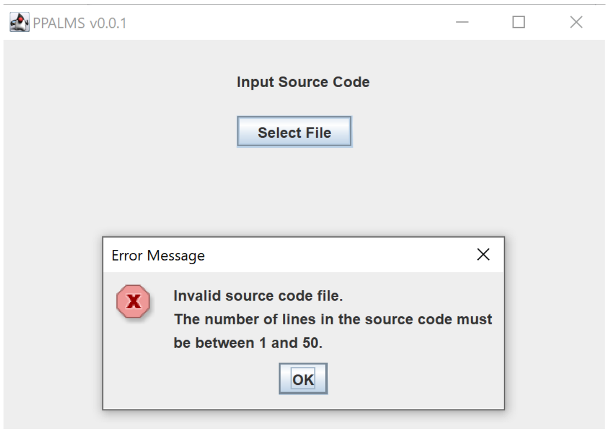
Expected Results: An error message should be displayed indicating that “Invalid File Extension”.

Test Data: File with no lines of code.

Dependencies: The PPALMS application has just been launched by the user and the CodeInputStrategy is displayed.

Empty Source Code File Selected	PPALMS v0.0.1
Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy.
Execution Report	Application window opens to code input as expected

<table>
<tr><td></td><td>
</td></tr>
<tr><td colspan="2"><b>Step 2: Click the Select File button</b></td></tr>
<tr><td>Expected Result</td><td>The PPALMS file system window has opened and the user is able to view the CodeInputStrategy.</td></tr>
<tr><td>Execution Report</td><td>File system window opens as expected

</td></tr>
<tr><td colspan="2"><b>Step 3: Select an invalid file (in this case, used empty_file.java)</b></td></tr>
<tr><td>Expected Result</td><td>The selected source code file appears in the File Name text field and is highlighted</td></tr>
</table>

Execution Report	Source code file appears in File Name text field  Look In: src consumer.c empty_file.java main.c overly_large_file.java producer.c File Name: empty_file.java Files of Type: All Files
Step 4: Click the "Open" button	
Expected Result	Invalid Source Code File Error Message will appear and "OK" can be clicked to restart the Input Source Code operation.
Execution Report	Invalid Source Code File Error Message appears.  Error Message Invalid source code file. The number of lines in the source code must be between 1 and 50.

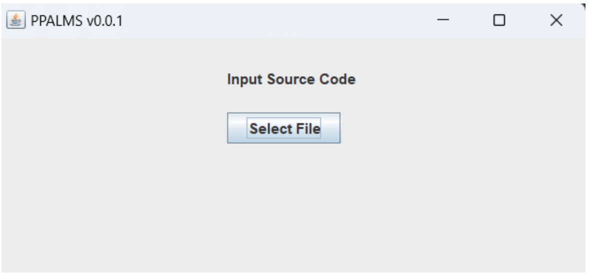
<u>Functional Test Case #5 Unsuccessful File Input - Source Code File with Too Many Lines Selected</u>

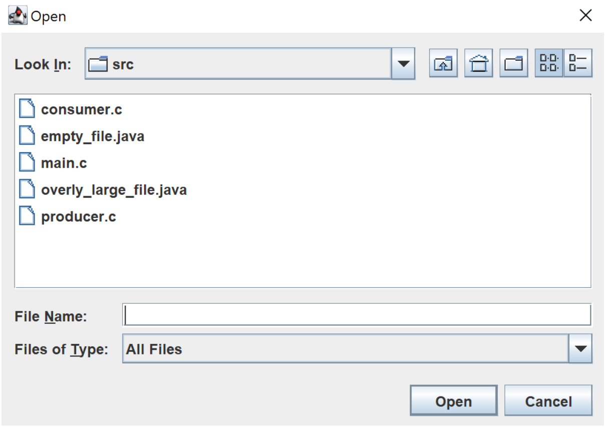
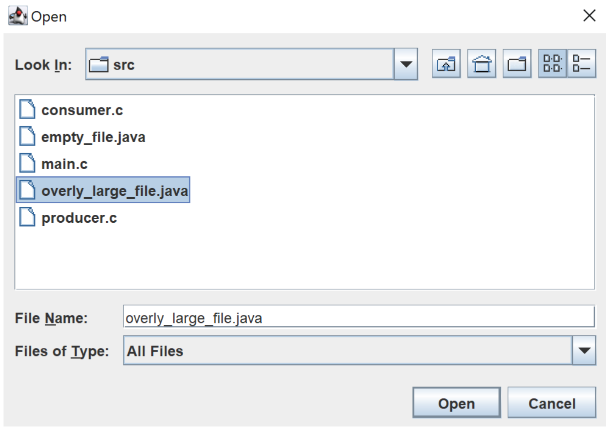
Purpose: This functional tests manually how the GUI responds to an unsuccessful file input where a source code file has too many lines in it.

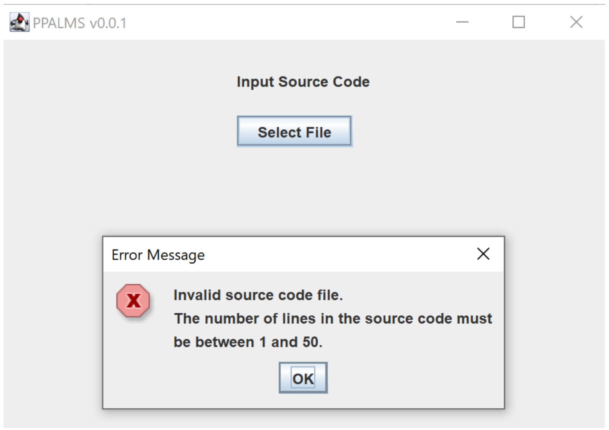
Owners: Anthony Narlock, Jaden Rodriguez, Shen Lua, Stephanie Ye

Expected Results: view strategy changed, reflected in the Graphical User Interface. An error message should be displayed indicating "Invalid File Extension"

Test Data: Input file with over 50 lines of code
Dependencies: The PPALMS application has just been launched by the user and the CodeInputStrategy is displayed

Source Code File with Too Many Lines Selected	PPALMS v0.0.1
Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy.
Execution Report	Application window opens to code input as expected 
Step 2: Click the Select File button	
Expected Result	The PPALMS file system window has opened and the user is able to view the CodeInputStrategy.

Execution Report	File system window opens as expected  Open consumer.c empty_file.java main.c overly_large_file.java producer.c Open Cancel Look In: src File Name: Files of Type: All Files
Step 3: Select an invalid file (in this case, used overly_large_file.java)	
Expected Result	The selected source code file appears in the File Name text field and is highlighted.
Execution Report	The source code file appears in File Name text field.  Open consumer.c empty_file.java main.c overly_large_file.java producer.c Open Cancel Look In: src File Name: overly_large_file.java Files of Type: All Files
Step 4: Click the "Open" button	
Expected Result	Invalid Source Code File Error Message will appear and "OK" can be clicked to restart the Input Source Code operation.
Execution Report	Invalid Source Code File Error Message appears.

	 Error Message Invalid source code file. The number of lines in the source code must be between 1 and 50.

Functional Test Case #6 LMSInputStrategy Confirm Button successfully changes view to ProblemInputStrategy

Purpose: Clicking on the LMSInputStrategy confirm button after selecting a target LMS and Problem Type, should redirect the user to the ProblemInputStrategy view whereby details of the problem can be provided and annotations made.

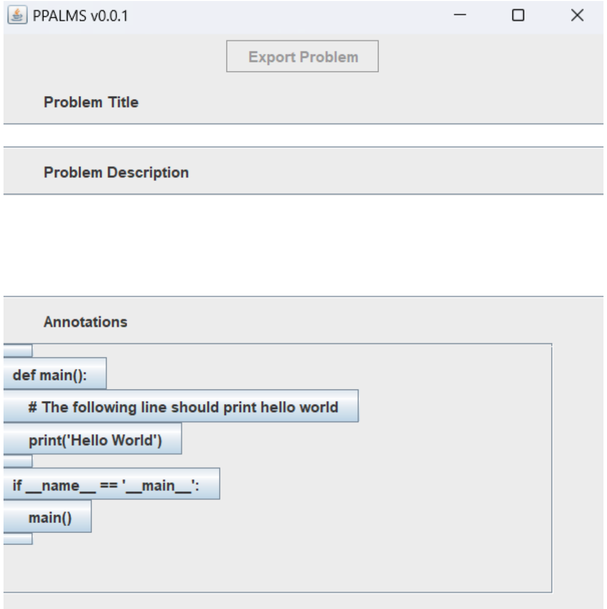
Owners: Shen Lua

Expected Results: The view strategy changes from LMSInputStrategy to ProblemInputStrategy

Test Data: N/A

Dependencies: The user must have selected a LMS target and Problem Type.

Confirm Button successfully changes view to ProblemInputStrategy	PPALMS v0.0.1
Step 1: Click on the Confirm Button	
Expected Result	LMSInputStrategy is updated to ProblemInputStrategy, which is demonstrated in the GUI.
Execution Report	ProblemInputStrategy view is displayed in the GUI.

	
	 PPALMS v0.0.1
	Export Problem
	Problem Title
	Problem Description
	Annotations
	def main():
	    # The following line should print hello world
	    print('Hello World')
	if __name__ == '__main__':
	    main()

Functional Test Case #7 Export Problem Failure due to already existing File

Purpose: Test manually that the GUI prompts the user with the correct error message upon detecting export directory file name conflict..

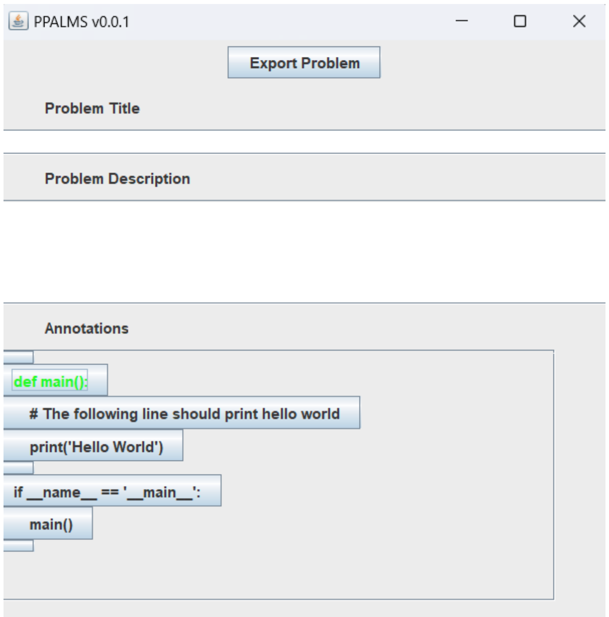
Owner: Jaden Rodriguez

Expected Results: An error message should be displayed indicating “A system error occurred which prevented the file from being made. Most likely, the file "problem.json" already exists, and you should move or delete it.”

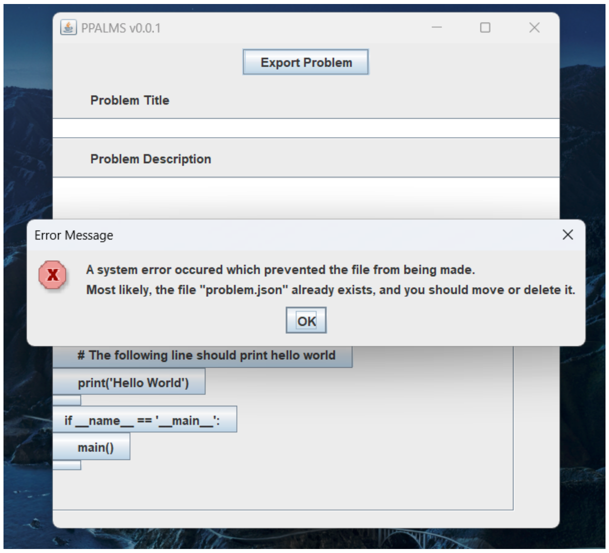
Test Data: GUI instance, file named “problem.json” in the export directory

Dependencies: The user must have finished configuring a valid PPALMS problem

Export Problem Failure due to already existing File	PPALMS v0.0.1
Step 1: Configure Valid PPALMS problem	
Expected Result	Export button should become clickable

Execution Report	 PPALMS v0.0.1
	Export Problem
	Problem Title
	Problem Description
	Annotationsdef main():
	# The following line should print hello world
	print('Hello World')
	if __name__ == '__main__':
	main()
	The ProblemInputStrategy view consists of a title and description field, and

Step 2: Click the export button	
Expected Result	An error message should be displayed indicating “A system error occurred which prevented the file from being made. Most likely, the file "problem.json" already exists, and you should move or delete it.”

Execution Report	An Error Dialog message is displayed.
	 PPALMS v0.0.1
	Export Problem
	Problem Title
	Problem Description
	Error Message
	OKA system error occured which prevented the file from being made.
	Most likely, the file "problem.json" already exists, and you should move or delete it.
	# The following line should print hello world
	print('Hello World')
	if __name__ == '__main__':
	main()

2.1.2 PpalmsGUI Unit Tests

Test Case #1 testUpdateViewStrategySuccess
Purpose: Tests whether the view strategy has changed from the default “CodeInputStrategy” to “LMSInputStrategy” in this testing scenario when updateViewStrategy method is invoked.
Owner: Shen Lua
Expected Results: view strategy changed, reflected in the Graphical User Interface.
Test Data: N/A
Dependencies: The PPALMS application has just been launched by the user. This indicates that the view strategy of the application will be on CodeInputStrategy

testUpdateViewStrategy	PPALMS v0.0.1
Step 1: Upload a valid SourceCode file.	
Expected Result	Once a valid SourceCode file has been uploaded, the view strategy shall change from CodeInputStrategy to LmsInputStrategy, then when the target lms and problem type are selected and upon clicking the confirmTargetLms button, the view will again change to the ProblemInputStrategy view.

Test Case #2 testProblemTitleDescriptionInput
Purpose: Tests whether both the Title and Description JTextFields are operational and accept user input.
Owner: Shen Lua
Expected Results: ability to type in both title and description fields upon clicking on them.
Test Data: N/A
Dependencies: The user has provided a valid SourceCode file, and has selected a target LMS and Problem Type and is in the ProblemInputStrategy view.

testUpdateViewStrategy	PPALMS v0.0.1
Step 1: Type in both the Title and Description fields	

Expected Result	The user is able to type in both the title and description fields without an error dialog.

Test Case #3 testProblemAnnotation

Purpose: Tests whether clicking on a line button in the ProblemInputStrategy view for annotation would result in the exportProblem button to be enabled.
Owner: Shen Lua
Expected Results: The exportProblem button will be enabled. The line button will turn text within it green in color.
Test Data: N/A
Dependencies: The user has provided a valid SourceCode file, and has selected a target LMS and Problem Type and is in the ProblemInputStrategy view.

testUpdateViewStrategy	PPALMS v0.0.1
Step 1: Click on a line button.	
Expected Result	The exportProblem button will be enabled. The line button will turn text within it green in color.

Test Case #4 testSetTargetLMSSuccess

Purpose: Tests the intended behavior of the LMSInputStrategy when no lms target is selected and when a lms target is selected from the lmsTargetComboBox.
Owner: Shen Lua
Expected Results: ProblemType ComboBox and confirmLMSTarget button are both disabled. Displayed as faded in the Graphical User Interface.
Test Data: N/A
Dependencies: A valid SourceCode file has been uploaded successfully, and the current view is the LMSInputStrategy.

testTargetLMSNoneSelected	PPALMS v0.0.1
Step 1: A Valid SourceCode file has been uploaded successfully.	
Expected Result	ProblemType ComboBox enabled when any of the target LMS options are selected. Otherwise, disabled.

Test Case #5 testProblemTypeSelection

Purpose: Tests the intended system behavior when the problem type is not selected and selected.
Owner: Shen Lua
Expected Results: The confirmLmsTarget button would remain disabled until a ProblemType is selected.
Test Data: N/A
Dependencies: A valid SourceCode file has been uploaded successfully, and the current view is the LMSInputStrategy, with an LMS target selected.

testProblemTypeSelection	PPALMS v0.0.1
Step 1: Don't select anything from the Problem Type ComboBox	
Expected Result	confirmLmsTarget button will remain disabled until a problem type is selected from the ProblemType ComboBox.

2.2 ViewStrategy Tests

The following subsections test the concrete implementations of the ViewStrategy. As specified in the design document, the ViewStrategy is an interface that physically shows the components on the screen to the user. Therefore, it is sufficient to test the concrete implementations of the ViewStrategy that are used in the PPALMS application.

2.2.1 CodeInputStrategy Tests

Test Case #1 testCodeInputStrategyAttributesNotNull

Purpose: Ensure that the CodeInputStrategy user interface panel components have been initialized.
Owner: Anthony Narlock
Expected Results: The codeInputButton, codeInputLabel, and fileChooser components are not null. This means that the user should be able to see these when going through the steps.
Test Data: Not applicable for a test related to graphical user interface.
Dependencies: The PPALMS application has just been launched by the user. This indicates that the view strategy of the application will be on CodeInputStrategy.
Note: This test serves the purpose of testing the initialization of components for the graphical user interface. This means that there will be no input cases. Therefore implying there are no edge cases for this test.

testCodeInputStrategyAttributesNotNull	PPALMS v0.0.1
Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to see the codeInputLabel, codeInputButton.

Test Case #2 testCodeInputStrategyComponentAttributes

Purpose: Ensure that the CodeInputStrategy interface components have correct visual properties.

Owner: Anthony Narlock

Expected Results: The codeInputButton has the correct text field value of "Select File" and the codeInputLabel has the correct text field value of "Input Source Code".

Test Data: Not applicable for a test related to graphical user interface.

Dependencies: The PPALMS application has just been launched by the user. This indicates that the view strategy of the application will be on CodeInputStrategy.

Note: This test serves the purpose of testing the initialization of components for the graphical user interface. This means that there will be no input cases. Therefore implying there are no edge cases for this test.

testCodeInputStrategyComponentAttributes	PPALMS v0.0.1
Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to see the codeInputLabel, codeInputButton. In addition, the user should see the label that visually says "Input Source Code" and the button visually says "Select File".

2.2.2 LMSInputStrategy Tests

Test Case #1 testAllLMSOptionsExist InLMSTargetComboBox

Purpose: To ensure all LMS target options are available for selection in the ComboBox.

Owner: Shen Lua

Expected Results: All LMS target results should be displayed upon the invocation by clicking on the LMSTargetComboBox.

Test Data: Not applicable for a test related to the Graphical User Interface
Dependencies: The user has already uploaded a valid source code file, which took place in the initial view strategy: “CodeInputStrategy”. The current view strategy should be the “LMSInputStrategy”.

testAllLMSOptionsExistInLMSTargetComboBox	PPALMS v0.0.1
Step 1: Click on the downward arrow of LMSTargetComboBox.	
Expected Result	Once a valid SourceCode file has been uploaded, a different view strategy is displayed : “LMSInputStrategy”, which consists of labels, ComboBoxes, and a confirm button. The user may then click on the downward arrow of the LMSTargetComboBox, and a dropdown list of LMS targets would be shown.

Test Case #2 testAllProblemTypesExistInProblemTypeComboBox
Purpose: To ensure all problem type options are available for selection in the ComboBox.
Owner: Shen Lua
Expected Results: All Problem Types should be displayed upon the invocation by clicking on the ProblemTypeComboBox.
Test Data: Not applicable for a test related to the Graphical User Interface.
Dependencies: The user has already uploaded a valid source code file, which took place in the initial view strategy: “CodeInputStrategy”. The user then selected a target LMS from the TargetLMSComboBox in the current view strategy “LMSInputStrategy”.

testAllProblemTypesExistInProblemTypeComboBox	PPALMS v0.0.1
Step 1: Click on the downward arrow of ProblemTypeComboBox	
Expected Result	Once a LMS target has been selected from the LMSTarget ComboBox, the user may then proceed by clicking on the downward arrow for the problemTypeComboBox and a list of problem types would be shown in the dropdown list.

Test Case #3 testControllerActionsSet

Purpose: To ensure default controller behavior, disables both ProblemType ComboBox and confirmLmsTargetButton until further action.

Owner: Shen Lua

Expected Results: ProblemType ComboBox and ConfirmLMSTargetButton are disabled by default

Test Data: Not applicable for a test related to the Graphical User Interface

Dependencies: The user has already uploaded a valid source code file, which took place in the initial view strategy: “CodeInputStrategy”. The current view strategy should be the “LMSInputStrategy”.

testControllerActionsSet	PPALMS v0.0.1
Step 1: Successfully uploaded a valid source code file.	
Expected Result	ProblemType ComboBox and ConfirmLMSTargetButton are disabled by default until prior actions are completed.

Test Case #4 testLmsTargetLabelExists

Purpose: To ensure that target LMS label exists so as to guide the user.

Owner: Shen Lua

Expected Results: target LMS label is displayed

Test Data: Not applicable for a test related to the Graphical User Interface

Dependencies: The user has already uploaded a valid source code file, which took place in the initial view strategy: “CodeInputStrategy”. The current view strategy should be the “LMSInputStrategy”.

testLmsTargetLabelExists	PPALMS v0.0.1
Step 1: Successfully uploaded a valid	

source code file.	
Expected Result	Once a valid SourceCode file has been uploaded, the “LMSInputStrategy” strategy view would be displayed to the user, which consists of the multiple components such as labels and ComboBoxes. One of the two labels is the LMSTarget label.

Test Case #5 testProblemTypeLabelExists

Purpose: To ensure that problem type labels are present for the user.

Owner: Shen Lua

Expected Results:

Test Data: Not applicable for a test related to the Graphical User Interface

Dependencies: The user has already uploaded a valid source code file, which took place in the initial view strategy: “CodeInputStrategy”. The current view strategy should be the “LMSInputStrategy”.

testProblemTypeLabelExists	PPALMS v0.0.1
Step 1: Successfully uploaded a valid source code	
Expected Result	Once a valid SourceCode file has been uploaded, the “LMSInputStrategy” strategy view would be displayed to the user, which consists of the multiple components such as labels and ComboBoxes. One of the two labels is the ProblemType label.

2.2.3 ProblemInputStrategy Tests

Test Case #1 testProblemInputStrategyAttributesNotNull

Purpose: Ensure that the ProblemInputStrategy user interface panel components have been initialized.

Owner: Anthony Narlock

Expected Results: The titleInputLabel, titleInputTextField, descriptionInputLabel, descriptionInputTextField, exportProblem button, annotationsLabel, and annotationPanel

components are not null. This means that the user should be able to see these when going through the steps.
Test Data: Not applicable for a test related to graphical user interface.
Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, and problem type.
Note: This test serves the purpose of testing the initialization of components for the graphical user interface. This means that there will be no input cases. Therefore implying there are no edge cases for this test.

testProblemInputStrategyAttributesNotNull	PPALMS v0.0.1
Steps for this test are done through satisfying the dependencies.	
Expected Result	The PPALMS application window has opened and the user is able to see the each of the ProblemInputStrategy components mentioned above.

Test Case #2 testProblemInputStrategyComponentAttributes
Purpose: Ensure that the ProblemInputStrategy interface components have correct visual properties.
Owner: Anthony Narlock
Expected Results: The titleInputLabel has the correct text value of “Problem Title”, titleInputTextField has the correct text value of the empty string, descriptionInputLabel has the correct text value of “Problem Description”, descriptionInputTextField has the correct text value of the empty string, annotationsLabel has the correct text value of “Annotations”, and the exportProblem button has the correct text value of “Export Problem”
Test Data: Not applicable for a test related to graphical user interface.
Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, and problem type.
Note: This test serves the purpose of testing the initialization of components for the graphical user interface. This means that there will be no input cases. Therefore implying there are no edge cases for this test.

testProblemInputStrategyComponentAttributes	PPALMS v0.0.1
Steps for this test are done through satisfying the dependencies.	

Expected Result	The PPALMS application window has opened and the user is able to see each component specified in this test along with their expected value.

2.3 PpalmsInputHandler Tests

Test Case #1 testProcessInputInvalidInput

Purpose: This tests if no valid input is given to the InputHandler, nothing is processed.

Owner: Anthony Narlock, Stephanie Ye

Expected Results: The boolean method processInput will return false if it reads in an invalid parameter to the function (either invalid JComponent or event).

Test Data: Passing a null JComponent and an invalid event called "test" into the processInput method, both of which are invalid.

Dependencies: The PPALMS application is running.

testProcessInputInvalidInput	PPALMS v0.0.1
Step 1: The PPALMS system is running.	
Expected Result	Upon receiving any kind of invalid input, we will expect the processInput method to return false.

Test Case #2 testProcessInputSourceCodeExtensionSuccessful

Purpose: This tests the case where a source file with an acceptable file extension is inputted and processed.

Owner: Stephanie Ye

Expected Results: The boolean method processInput will return true if a valid file extension type is inputted with the "sourceCodeExtension" event

Test Data: JTextField source code file names: *test.py, test.java, test.cpp, test.c, and test.cc*

Dependencies: The PPALMS application is running.

testProcessInputSourceCodeExtensionSuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into processInput	

Expected Result	We then expect the switch statement inside of the processInput method to go into the “sourceCodeExtension" branch and each test data file type passed in will be validated. Since they are valid file types, we expect the switch statement to return true.

Test Case #3 testProcessInputSourceCodeExtensionUnsuccessful

Purpose: This tests the case where a source file with an unacceptable file extension is inputted and therefore not processed

Owner: Stephanie Ye

Expected Results: The boolean method processInput will return false if an invalid file extension type is inputted with the “sourceCodeExtension” event

Test Data: JTextField source code file names: *test.txt, test.png, test.mp4, and test.ml*

Dependencies: The PPALMS application is running and the view is on the CodeInputStrategy.

testProcessInputSourceCodeExtensionUnsuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the invalidTest Data into processInput	
Expected Result	We expect the switch statement inside of the processInput method to go into the “sourceCodeExtension” branch and each test data file type passed in will be validated. Since they are invalid file types, we expect the validateCodeInput to return false and therefore break out of the if statement and switch case to return false.

Test Case #4 testProcessInputSourceCodeInputUnsuccessful

Purpose: This tests if no source code has been inputted to the application and therefore nothing is processed

Owner: Stephanie Ye

Expected Results: The boolean method processInput will return false when it reaches the break in the switch case for “sourceCode” event.

Test Data: JTextField source code that is null

Dependencies: The PPALMS application is running and the view is on the CodeInputStrategy.

testProcessInputSourceCodeInputUnsuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the invalidTest Data into processInput	
Expected Result	We expect the switch statement in the processInput method to go into the "sourceCode" branch with a null JTextField passed in. We expect setSourceCode to set the source code to null and then will return false.

Test Case #5 testProcessInputLmsTargetInputSuccessful

Purpose: This tests the case where a valid target LMS is inputted and processed

Owner: Stephanie Ye

Expected Results: The boolean method processInput will return true if a valid target LMS is inputted with the "lmsTarget" event.

Test Data: JComboBox with the selection of: *Canvas, D2L, Absorb, Matrix, and Talent*

Dependencies: The PPALMS application is running and the view is on the LMSInputStrategy.

testProcessInputLmsTargetInputSuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into processInput	
Expected Result	We then expect the switch statement in the processInput method to go into the "lmsTarget" branch and JComboBox selections of LMS are passed in. We expect setLmsTarget to set to the LMS selection passed in and since it is a valid LMS, we expect it to return true.

Test Case #6 testProcessInputLmsTargetInputUnsuccessful

Purpose: This tests if an invalid target LMS is inputted and therefore not processed.

Owner: Stephanie Ye

Expected Results: The boolean method processInput will return false if an invalid target LMS is inputted with the "lmsTarget" event.

Test Data: JComboBox with the selection at the 0 index (this is just Expand and not an LMS)
Dependencies: The PPALMS application is running and the view is on the LMSInputStrategy.

testProcessInputLmsTargetInputUnsuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the invalid Test Data into processInput	
Expected Result	We then expect the switch statement in the processInput method to go into the “lmsTarget "branch and JComboBox selection of the 0th index item of the LMS combo box is passed in. We expect the selection to be caught by the IllegalArgumentException since the 0th index is Expand and not a valid LMS type and therefore it will return false.

<u>Test Case #7 testProcessInputProblemTypeInputSuccessful</u>
Purpose: This tests if a valid problem type is inputted and processed
Owner: Stephanie Ye
Expected Results: The boolean method processInput will return true if a valid problem type is inputted with the “problemType” event.
Test Data: JComboBox with the selection of: *Ordering (for now only implemented ordering type problems, but in the next version we will have Fill in Blank and Multiple Choice problem types)*
Dependencies: The PPALMS application is running and the view is on the LMSInputStrategy.

testProcessInputProblemTypeInputSuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into processInput	
Expected Result	We then expect the switch statement in the processInput method to go into the “problemType” branch and JComboBox selection of Ordering problem type is passed in. We expect setProblemType to set to the problem type selection passed in and since it is a valid problem type, we expect it to return true.

Test Case #8 testProcessInputProblemTypeInputUnsuccessful

Purpose: This tests if an invalid problem type is inputted and therefore not processed

Owner: Stephanie Ye

Expected Results: The boolean method processInput will return false if an invalid problem type is inputted with the “problemType” event.

Test Data: JComboBox with the selection at the 0 index (this is just Expand and not a problem type)

Dependencies: The PPALMS application is running and the view is on the LMSInputStrategy.

testProcessInputProblemTypeInputUnsuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the invalid Test Data into processInput	
Expected Result	The PPALMS application window has opened and the user is able to view the LMSInputStragey. We then expect the switch statement to go into the “problemType " branch and JComboBox selection of the 0th index item of the problem type combo box is passed in. We expect the selection to be caught by the IllegalArgumentException since the 0th index is Expand and not a valid problem type and therefore it will return false.

Test Case #9 testProcessInputProblemTtileInputSuccessful

Purpose: This tests if a problem title is inputted and processed.

Owner: Stephanie Ye

Expected Results: The boolean method processInput for the case problemTitle will always return true because even when passing in a JTextField that is null, when it does JComponenet.getText(), it processes it as an empty string "" which is valid

Test Data: JTextField problem titles: *“test title” and ““* with the “problemTitle” event

Dependencies: The PPALMS application is running and the view is on the ProblemInputStrategy.

testProcessInputProblemTtileInputSuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into	

processInput	
Expected Result	The PPALMS application window has opened and the user is able to view the ProblemInputStrategy. We then expect the switch statement to go into the “problemTitle " branch and the JTextField problem titles are passed in. We expect SetTitle to set to the problem title passed in and since it is a valid title, we expect it to return true.

<u>Test Case #10 testProcessInputProblemDescriptionInputSuccessful</u>

Purpose: This tests if a problem description is inputted and processed.

Owner: Stephanie Ye

Expected Results: The boolean method processInput for the case problemDescription will always return true because even when passing in a JTextArea that is null, when it does JComponenet.getText(), it processes it as an empty string "" which is valid

Test Data: JTextArea problem descriptions: *"test description: This is a PPALMS Problem" and ““* with the “problemDescription” event

Dependencies: The PPALMS application is running and the view is on the ProblemInputStrategy.

testProcessInputProblemDescriptionInputSuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into processInput	
Expected Result	The PPALMS application window has opened and the user is able to view the ProblemInputStrategy. We then expect the switch statement to go into the “problemDescription " branch and the JTextArea problem descriptions are passed in. We expect setDescription to set to the problem description passed in and since it is a valid description, we expect it to return true.

<u>Test Case #11 testProcessInputExportProblemInputSuccessful</u>

Purpose: This tests if the export problem is processed given an input PPALMS problem.

Owner: Stephanie Ye
Expected Results: The exportPpalmsProblem is expected to return true if all the necessary inputs for a PPALMS problem is given as well as the "exportProblem " event.
Test Data: A sample problem with lines of code, a title and description, a target LMS of Canvas, a problem type of Ordering, and annotations.
Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy. The file system is cleared of any conflicting file names if they exist (usually problem.json if export problem has been used before)

testProcessInputExportProblemInputSuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the problem attributes which is the valid Test Data into processInput	
Expected Result	The PPALMS application window has opened and the user is able to view the ProblemInputStrategy. We then expect source code and its file extension type to be passed through the switch statement and validated, then the target LMS, then problem type, then title and description, then annotations before the switch statement goes into the "exportProblem" branch. All the previous problem attributes are then passed to the exportPpalmsProblem method in the PpalmsHandler and is expected to return true because the problem attributes passed in are all valid and are able to create permuted problems, as well as because the file system is cleared of any conflicting file names if they exist.

Test Case #12 testProcessInputSourceCodeLinesInputSuccessful
Purpose: This tests if user input source code lines are processed
Owner: Stephanie Ye
Expected Results: The boolean method processInput for source code lines returns true if the source code has a length between 1-50.
Test Data: A list of Strings which represent the lines in source code and is comprised of: *"void function()", "function(variable)", and "x=null"*
Dependencies: The PPALMS application is running and the view is on CodeInputStrategy

testProcessInputSourceCodeLinesInputSuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into processInput	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy. We then expect the lines of source code to be passed into the processInput function and we expect setSourceCodeLines to set to the source code lines passed in. Since they are valid source code lines (number of lines between 1-50), we expect it to return true.

Test Case #13 testProcessInputSourceCodeLinesInputUnsuccessful

Purpose: This tests if user inputs invalid source code and so it is not processed

Owner: Stephanie Ye

Expected Results: The boolean method processInput for source code lines returns false if the source code has a length not in between 1-50.

Test Data: A list of Strings which represent the lines in source code. One is an emptyList and the other is comprised of 51 “lines” of code that are all “test”

Dependencies: The PPALMS application is running and the view is on CodeInputStrategy

testProcessInputSourceCodeLinesInputUnsuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into processInput	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy. We then expect the lines of source code to be passed into the processInput function. We expect the function to return false because an empty source code List and a source code List with 51 lines is passed in which are invalid source code lines because it isn’t between 1-50 lines

Test Case #14 testProcessInputAddAnnotationSuccessful

Purpose: This tests if valid user input annotations are processed

Owner: Stephanie Ye

Expected Results: The boolean method processInput for annotations returns true for successful adding annotation inputs

Test Data: Index of the line to annotate as well as the "*addAnnotation*" event

Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy

testProcessInputAddAnnotationSuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into processInput	
Expected Result	The PPALMS application window has opened and the user is able to view the LMSInputStrategy. We then expect the line index for annotations to be passed into the processInput function and we expect to go into the switch case "addAnnotation" where the getAnnotation function either instantiates an annotation that is initially null or returns the annotation. We always expect processInput to return true given that the "addAnnotation" event is passed in.

Test Case #15 testProcessInputAddAnnotationUnsuccessful

Purpose: This tests if event parameter is not valid for processInput of annotations (i.e. not addAnnotation)

Owner: Stephanie Ye

Expected Results: The boolean method processInput for annotations returns false if an invalid event is passed into the method.

Test Data: Index of the line to annotate as well as an invalid event called "*test*"

Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy

testProcessInputAddAnnotationUnsuccessful	PPALMS v0.0.1
Step 1: Given the dependencies, the system passes in the valid Test Data into processInput	

Expected Result	The PPALMS application window has opened and the user is able to view the LMSInputStrategy. We then expect the line index for annotations to be passed into the processInput function but because an invalid event “test” is passed in, none of the switch cases are triggered so we expect it to fall through and return false

2.4 PpalmsLogicHandler Tests

Test Case #1 testValidateCodeInputSuccess

Purpose: Ensure validity of a user’s source code file. This test executes the validateCodeInput method in the PpalmsLogicHandler.
Owner: Anthony Narlock
Expected Results: The validateCodeInput method in the PpalmsLogicHandler will return true for valid source code file extensions: py, java, cpp, c, and cc.
Test Data: Source code path with file names: *test.py, test.java, test.cpp, test.c, and test.cc.*
Dependencies: The PPALMS application has just been launched by the user and the user has inputted their source code file.

testValidateCodeInputSuccess	PPALMS v0.0.1
Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy.
Step 2: Select source code file containing valid extension from the file chooser dialog menu.	
Expected Result	The PPALMS application will successfully validate the source code. The view will change to the LMSInputStrategy.

Test Case #2 testValidateCodeInputNoCode

Purpose: Ensure validity of a user's source code file under the condition of failure to select a source code file. This can happen if the user selects cancel on the file chooser dialog. This test executes the validateCodeInput method in the PpalmsLogicHandler.
Owner: Anthony Narlock
Expected Results: The validateCodeInput method in the PpalmsLogicHandler will return false for no source code provided.
Test Data: A "null" attribute. Since no source code is inputted for the problem, the attribute of the problem will be null.
Dependencies: The PPALMS application has just been launched by the user and the user has inputted their source code file.

testValidateCodeInputNoCode	PPALMS v0.0.1
Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy.
Step 2: Selects source file button, but selects cancel on the chooser dialog.	
Expected Result	The PPALMS application will unsuccessfully validate the source code. An error message will be displayed on the screen to indicate invalid file.

Test Case #3 testValidateCodeInputInvalidExtension

Purpose: Ensure validity of a user's source code file under the condition of failure to select a source code file. This can happen if the user selects a file that contains an unsupported file extension. This test executes the validateCodeInput method in the PpalmsLogicHandler.
Owner: Anthony Narlock
Expected Results: The validateCodeInput method in the PpalmsLogicHandler will return false for an invalid file extension.
Test Data: Source code path with file names: *test.txt, test.png, test.mp4, test.ml.*
Dependencies: The PPALMS application has just been launched by the user and the user has inputted their source code file.

testValidateCodeInputNoCode	PPALMS v0.0.1

Step 1: Open the PPALMS application JAR file.	
Expected Result	The PPALMS application window has opened and the user is able to view the CodeInputStrategy.
Step 2: Select source code file containing invalid extension from the file chooser dialog menu.	
Expected Result	The PPALMS application will unsuccessfully validate the source code. An error message will be displayed on the screen to indicate invalid file.

<u>Test Case #4 testValidateTitleInputSuccess</u>

Purpose: Ensure that the PPALMS application properly validates title input in regards to the requirements specification. That is, the user can choose to leave the field blank and not indicate a title since it is optional.

Owner: Anthony Narlock

Expected Results: The validateTitleInput method in the PpalmsLogicHandler will return true for valid title input.

Test Data: Title attributes of type string “Test” and “” (the empty string).

Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, and problem type.

testValidateTitleInputSuccess	PPALMS v0.0.1
Step 1: Write a title inside of the title text field - or choose to leave blank for no title.	
Expected Result	The PPALMS application’s PpalmsLogicHandler will return true for any text field value due to the nature of the requirements for title input.

<u>Test Case #5 testValidateTitleInputFailure</u>

Purpose: Ensure that the PPALMS application will return failure under an unexpected condition that does not initialize the title input text field in the ProblemInputStrategy.

Owner: Anthony Narlock

Expected Results: The validateTitleInput method in the PpalmsLogicHandler will return false since the title object is null.
Test Data: Title attribute set to null.
Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, and problem type.

testValidateTitleInputFailure	PPALMS v0.0.1
Step 1: Write a title inside of the title text field - or choose to leave blank for no title.	
Expected Result	The PPALMS application's PpalmsLogicHandler will return false and indicate an error dialog displaying that an unexpected error occurred due to the text field object being null.

<u>Test Case #6 testValidateDescInputSuccess</u>
Purpose: Ensure that the PPALMS application properly validates description input in regards to the requirements specification. That is, the user can choose to leave the field blank and not indicate a description since it is optional.
Owner: Anthony Narlock
Expected Results: The validateDescInput method in the PpalmsLogicHandler will return true for valid title input.
Test Data: Title attributes of type string "Test" and "" (the empty string).
Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, and problem type.

testValidateDescInputSuccess	PPALMS v0.0.1
Step 1: Write a description inside of the description text field - or choose to leave blank for no title.	
Expected Result	The PPALMS application's PpalmsLogicHandler will return true for any text field value due to the nature of the requirements for description input.

<u>Test Case #7 testValidateDescInputFailure</u>

Purpose: Ensure that the PPALMS application will return failure under an unexpected condition that does not initialize the description input text field in the ProblemInputStrategy.
Owner: Anthony Narlock
Expected Results: The validateDescInput method in the PpalmsLogicHandler will return false since the title object is null.
Test Data: Description attribute set to null.
Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, and problem type.

testValidateTitleInputFailure	PPALMS v0.0.1
Step 1: Write a description inside of the description text field - or choose to leave blank for no title.	
Expected Result	The PPALMS application's PpalmsLogicHandler will return false and indicate an error dialog displaying that an unexpected error occurred due to the text field object being null.

Test Case #8 testSetAnnotations
Purpose: Given an input selection of annotations for an ordering problem. The annotations will come in the form of user-selected lines. Upon selecting each line, the PPALMS application will validate the selected annotations by sorting them. This allows the user to select lines of code in their problem out of order and still receive their expected result.
Owner: Anthony Narlock
Expected Results: Upon calling the PpalmsLogicHandler's setAnnotations method, the source code lines of the indicated problem will be in order based on the original source code provided by the user.
Test Data:

- A sample "Hello world" program written in Python where lines are selected in-order.
- A sample "Hello world" program written in Python where lines are not selected in-order.
- A sample "Hello world" program written in Python where the user only selects a minimal amount of lines. Indicating that we would expect our source code lines to hold proper size and not add lines that were not selected.

Note: *The test data represented here is sufficient enough to cover all possibilities of input for annotations. The user will not be able to proceed to exporting their problem unless an annotation line is implemented. As of PPALMS v0.0.1, the user can not unselect a problem annotation. Therefore, we need not worry about receiving no annotation input.*

Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, problem type, title, description, annotations.

testValidateTestSetAnnotations	PPALMS v0.0.1
Step 1: Given the dependencies, the user selects the export button from the ProblemInputStrategy	
Expected Result	The PPALMS application's PpalmsLogicHandler will be called inside of the export problem process. We will expect the setAnnotations method to return true for all test data used.

Test Case #9 testCreatePermutations

Purpose: Given an input PPALMS problem. The problem will have a list of annotations with a correct order, but different answer orders to present to students will not have been generated. The PPALMS application will generate a list of permuted problems meeting the length constraints in the design document so that several annotation orders can be listed in the exported problem.

Owner: Jaden Rodriguez

Expected Results: Upon calling the PpalmsLogicHandler's createPermutations method, a list of annotation-permuted copies of the indicated input problem will be generated. The list will meet the length constraints of being between length 1 and 30 inclusive, and will consist of unique and correct permutations.

Test Data:

- A sample problem where one line is selected (Lower bound of design constraints).
- A sample problem where multiple lines with fewer permutations than the length limit (30) are selected (2, 3, 4)
- A sample problem where multiple lines with more permutations than the length limit (30) are selected (5 and up)
- A sample problem where 50 lines are selected (Upper bound of design constraints)
- Each viable length was tested because it was feasible and provided larger coverage

Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, problem type, title, description, and annotations.

testCreatePermutations	PPALMS v0.0.1
Step 1: Given the dependencies, the user	

clicks the export button	
Expected Result	The PPALMS application's PpalmsLogicHandler will be called inside of the export problem process. We will expect the setAnnotations method to return true for all test data used. This provides an initial correctly permuted problem for the createPermutations method to use as a reference. We will expect it to return a maximal unique problem permutation list which satisfies the design constraints, such as being in the correct length domain.

Test Case #10 testExportPpalmsProblem

Purpose: Given an input PPALMS problem. The problem will have a list of annotations with a correct order. The PPALMS application will generate a list of permuted problems, and successfully export it to the local file system in a JSON format for ease-of-use. This allows an LMS or professor to parse the completed problem and retrieve its title, description, type, target LMS, correct annotations, and already generated permutations.

Owner: Jaden Rodriguez

Expected Results: Upon calling the PpalmsLogicHandler's exportPpalmsProblem method, a file specifying the completed and permuted problem in a JSON format will be generated. Because it uses the other methods, it will meet the length constraints and such. The JSON is expected to always have a field for each of: title, description, type, target LMS, correct annotations, and generated annotation permutations. In this test, we are testing for the success case, so it is ensured that there are no file conflicts, hence we expect the method to return true

Test Data:

- A sample problem with title and description, and a target LMS of Canvas
- A sample problem without a title nor a description, and a target LMS of D2L
- In both cases, the file system is cleared of any conflicting file names if they exist

Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, problem type, title, description, and annotations. This test also relies on the standard File I/O and JSON libraries implemented in Java.

testExportPpalmsProblem	PPALMS v0.0.1
Step 1: Given the dependencies, the user clicks the export button	
Expected Result	PpalmsLogicHandler will be called inside of

	the export problem process. We will expect the setAnnotations method to return true and the createPermutations method to make a list of permuted problems. To finish the process, this method will parse the original problem and the permutations list. After finding no filename conflicts, it will write their contents into a JSON file and return true upon success.

Test Case #11 testExportPpalmsProblemFail

Purpose: Given an input PPALMS problem. The PPALMS application will generate a list of permuted problems, and attempt to export it to the local file system in a JSON format for ease-of-use. It may find that there exists a file with a conflicting name, in which case it should return false early to signal an error prompt to the view. This allows the user to adjust their file system accordingly.

Owner: Jaden Rodriguez

Expected Results: Upon calling the PpalmsLogicHandler's exportPpalmsProblem method, a file specifying the completed and permuted problem in a JSON format will be generated. Due to a name conflict, it will not be written to successfully, and will flag the view to prompt the user with an error message. In this test, we are testing for the fail case, so it is ensured that there is always a file conflict, hence we expect the method to return false.

Test Data:

- A sample problem with initialized fields, and a file system with a conflicting file name in the export directory

Dependencies: The PPALMS application is running and the view is on ProblemInputStrategy - indicating the user has already successfully entered in source code, provided an LMS target, problem type, title, description, and annotations. This test also relies on the standard File I/O and JSON libraries implemented in Java.

testCreatePermutations	PPALMS v0.0.1
Step 1: Given the dependencies, the user clicks the export button	
Expected Result	PpalmsLogicHandler will be called inside of the export problem process. We will expect the exportPpalmsProblem to find filename conflicts. Hence, it won't create an output, and will return false. This will flag the view to produce an error message.

3. Unit Test Execution Report

This section provides an extensive overview of the execution reports for each test class listed in section 2. The reports indicate whether specific tests have passed or failed over several execution trials. The results of the trail runs are color-coded in green (PASS) and red (FAIL) for enhanced readability. The total number of unit tests that were conducted for PPALMS version 0.0.1 were 40. The number of tests per class are indicated with each of their tests. During our testing, every one of the unit tests we conducted in our application passed. For each of our tests, we ran a total of 276 executions, resulting in 100% passed executions and 0% failed executions.

3.1 PpalmsGui Test Execution Report

Test Case #1: testUpdateViewStrategySuccess **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testUpdateViewStrategySuccess - LMSInputStrategy - Execution #1	Nov. 23th, 2022	PASS
testUpdateViewStrategySuccess - LMSInputStrategy - Execution #2	Nov. 23th, 2022	PASS
testUpdateViewStrategySuccess - LMSInputStrategy - Execution #3	Nov. 23th, 2022	PASS
testUpdateViewStrategySuccess - ProblemInputStrategy -Execution #1	Nov. 24th, 2022	PASS
testUpdateViewStrategySuccess - ProblemInputStrategy - Execution #2	Nov. 24th, 2022	PASS
testUpdateViewStrategySuccess - ProblemInputStrategy - Execution #3	Nov. 24th, 2022	PASS

Test Case #2: testProblemTitleDescriptionInput **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProblemTitleDescriptionInput - Execution #1	Nov. 23th, 2022	PASS
testProblemTitleDescriptionInput - Execution #2	Nov. 23th, 2022	PASS
testProblemTitleDescriptionInput - Execution #3	Nov. 23th, 2022	PASS

testProblemTitleDescriptionInput - Execution #4	Nov. 24th, 2022	PASS
testProblemTitleDescriptionInput - Execution #5	Nov. 24th, 2022	PASS
testProblemTitleDescriptionInput - Execution #6	Nov. 24th, 2022	PASS

Test Case #3: testProblemAnnotation **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProblemAnnotation - Execution #1	Nov. 23th, 2022	PASS
testProblemAnnotation - Execution #2	Nov. 23th, 2022	PASS
testProblemAnnotation - Execution #3	Nov. 23th, 2022	PASS
testProblemAnnotation - Execution #4	Nov. 24th, 2022	PASS
testProblemAnnotation - Execution #5	Nov. 24th, 2022	PASS
testProblemAnnotation - Execution #6	Nov. 24th, 2022	PASS

Test Case #4: testSetTargetLMSSuccess **18/18 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testSetTargetLMSSuccess - Expand - Execution #1	Nov. 23th, 2022	PASS
testSetTargetLMSSuccess - Expand - Execution #2	Nov. 23th, 2022	PASS
testSetTargetLMSSuccess - Expand - Execution #3	Nov. 23th, 2022	PASS
testSetTargetLMSSuccess - Canvas - Execution #1	Nov. 24th, 2022	PASS
testSetTargetLMSSuccess - Canvas - Execution #2	Nov. 24th, 2022	PASS
testSetTargetLMSSuccess - Canvas - Execution #3	Nov. 24th, 2022	PASS
testSetTargetLMSSuccess - D2L - Execution #1	Nov. 25th, 2022	PASS
testSetTargetLMSSuccess - D2L - Execution #2	Nov. 25th, 2022	PASS
testSetTargetLMSSuccess - D2L - Execution #3	Nov. 25th, 2022	PASS
testSetTargetLMSSuccess - Absorb - Execution #1	Nov. 26th, 2022	PASS

testSetTargetLMSSuccess - Absorb - Execution #2	Nov. 26th, 2022	PASS
testSetTargetLMSSuccess - Absorb - Execution #3	Nov. 26th, 2022	PASS
testSetTargetLMSSuccess - Matrix - Execution #1	Nov. 27th, 2022	PASS
testSetTargetLMSSuccess - Matrix - Execution #2	Nov. 27th, 2022	PASS
testSetTargetLMSSuccess - Matrix - Execution #3	Nov. 27th, 2022	PASS
testSetTargetLMSSuccess - Talent - Execution #1	Nov. 28th, 2022	PASS
testSetTargetLMSSuccess - Talent - Execution #2	Nov. 28th, 2022	PASS
testSetTargetLMSSuccess - Talent - Execution #3	Nov. 28th, 2022	PASS

<u>Test Case #5:</u> testProblemTypeSelection **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProblemTypeNotSelected - Expand - Execution #1	Nov. 23th, 2022	PASS
testProblemTypeNotSelected - Expand - Execution #2	Nov. 24th, 2022	PASS
testProblemTypeNotSelected - Expand - Execution #3	Nov. 25th, 2022	PASS
testProblemTypeNotSelected - Ordering - Execution #1	Nov. 26th, 2022	PASS
testProblemTypeNotSelected - Ordering - Execution #2	Nov. 27th, 2022	PASS
testProblemTypeNotSelected - Ordering - Execution #3	Nov. 28th, 2022	PASS

3.2 ViewStrategy Test Execution Report

The following subsections display the test execution report for the concrete implementations of the ViewStrategy interface. As stated previously, there are no test cases for the interface due to the nature of abstraction.

3.2.1 CodeInputStrategy Test Execution Report

<u>Test Case #1:</u> testCodeInputStrategyAttributesNotNull **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testCodeInputStrategyAttributesNotNull - Execution #1	Nov. 20th, 2022	PASS

testCodeInputStrategyAttributesNotNull - Execution #2	Nov. 21st, 2022	PASS
testCodeInputStrategyAttributesNotNull - Execution #3	Nov. 22nd, 2022	PASS
testCodeInputStrategyAttributesNotNull - Execution #4	Nov. 25th, 2022	PASS
testCodeInputStrategyAttributesNotNull - Execution #5	Nov. 26th, 2022	PASS
testCodeInputStrategyAttributesNotNull - Execution #6	Nov. 27th, 2022	PASS

Test Case #2: testCodeInputStrategyComponentAttributes **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testCodeInputStrategyComponentAttributes - Execution #1	Nov. 20th, 2022	PASS
testCodeInputStrategyComponentAttributes - Execution #2	Nov. 21st, 2022	PASS
testCodeInputStrategyComponentAttributes - Execution #3	Nov. 22nd, 2022	PASS
testCodeInputStrategyComponentAttributes - Execution #4	Nov. 25th, 2022	PASS
testCodeInputStrategyComponentAttributes - Execution #5	Nov. 26th, 2022	PASS
testCodeInputStrategyComponentAttributes - Execution #6	Nov. 27th, 2022	PASS

3.2.2 LMSInputStrategy Test Execution Report

Test Case #1: testAllLMSOptionsExistInLMSTargetComboBox **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testAllLMSOptionsExistInLMSTargetComboBox - Execution #1	Nov. 23th, 2022	PASS
testAllLMSOptionsExistInLMSTargetComboBox - Execution #2	Nov. 24th, 2022	PASS
testAllLMSOptionsExistInLMSTargetComboBox - Execution #3	Nov. 25th, 2022	PASS
testAllLMSOptionsExistInLMSTargetComboBox - Execution #4	Nov. 26th, 2022	PASS
testAllLMSOptionsExistInLMSTargetComboBox - Execution #5	Nov. 27th, 2022	PASS

testAllLMSOptionsExistInLMSTargetComboBox - Execution #6	Nov. 28th, 2022	PASS

Test Case #2: testAllProblemTypesExistInProblemTypeComboBox **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testAllProblemTypesExistInProblemTypeComboBox - Execution #1	Nov. 23th, 2022	PASS
testAllProblemTypesExistInProblemTypeComboBox - Execution #2	Nov. 24th, 2022	PASS
testAllProblemTypesExistInProblemTypeComboBox - Execution #3	Nov. 25th, 2022	PASS
testAllProblemTypesExistInProblemTypeComboBox - Execution #4	Nov. 26th, 2022	PASS
testAllProblemTypesExistInProblemTypeComboBox - Execution #5	Nov. 27th, 2022	PASS
testAllProblemTypesExistInProblemTypeComboBox - Execution #6	Nov. 28th, 2022	PASS

Test Case #3: testControllerActionsSet **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testControllerActionsSet - Execution #1	Nov. 23th, 2022	PASS
testControllerActionsSet - Execution #2	Nov. 24th, 2022	PASS
testControllerActionsSet - Execution #3	Nov. 25th, 2022	PASS
testControllerActionsSet - Execution #4	Nov. 26th, 2022	PASS
testControllerActionsSet - Execution #5	Nov. 27th, 2022	PASS
testControllerActionsSet - Execution #6	Nov. 28th, 2022	PASS

Test Case #4: testLmsTargetLabelExists	**Date of Test**	PASS or

6/6 PASSED Execution Trials	**Execution**	FAIL
testLmsTargetLabelExists - Execution #1	Nov. 23th, 2022	PASS
testLmsTargetLabelExists - Execution #2	Nov. 24th, 2022	PASS
testLmsTargetLabelExists - Execution #3	Nov. 25th, 2022	PASS
testLmsTargetLabelExists - Execution #4	Nov. 26th, 2022	PASS
testLmsTargetLabelExists - Execution #5	Nov. 27th, 2022	PASS
testLmsTargetLabelExists - Execution #6	Nov. 28th, 2022	PASS

Test Case #5: testProblemTypeLabelExists **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProblemTypeLabelExists - Execution #1	Nov. 23th, 2022	PASS
testProblemTypeLabelExists - Execution #2	Nov. 24th, 2022	PASS
testProblemTypeLabelExists - Execution #3	Nov. 25th, 2022	PASS
testProblemTypeLabelExists - Execution #4	Nov. 26th, 2022	PASS
testProblemTypeLabelExists - Execution #5	Nov. 27th, 2022	PASS
testLmsTargetLabelExists - Execution #6	Nov. 28th, 2022	PASS

3.2.3 ProblemInputStrategy Test Execution Report

Test Case #1: testProblemInputStrategyAttributesNotNull **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProblemInputStrategyAttributesNotNull - Execution #1	Nov. 20th, 2022	PASS
testProblemInputStrategyAttributesNotNull - Execution #2	Nov. 21st, 2022	PASS
testProblemInputStrategyAttributesNotNull - Execution #3	Nov. 22nd, 2022	PASS
testProblemInputStrategyAttributesNotNull - Execution #4	Nov. 25th, 2022	PASS
testProblemInputStrategyAttributesNotNull - Execution #5	Nov. 26th, 2022	PASS

testProblemInputStrategyAttributesNotNull - Execution #6	Nov. 27th, 2022	PASS

Test Case #2: testProblemInputStrategyComponentAttributes **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProblemInputStrategyComponentAttributes - Exc. #1	Nov. 20th, 2022	PASS
testProblemInputStrategyComponentAttributes - Exc. #2	Nov. 21st, 2022	PASS
testProblemInputStrategyComponentAttributes - Exc. #3	Nov. 22nd, 2022	PASS
testProblemInputStrategyComponentAttributes - Exc. #4	Nov. 25th, 2022	PASS
testProblemInputStrategyComponentAttributes - Exc. #5	Nov. 26th, 2022	PASS
testProblemInputStrategyComponentAttributes - Exc. #6	Nov. 27th, 2022	PASS

3.3 PpalmsInputHandler Test Execution Report

Test Case #1: testProcessInputInvalidInput **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputInvalidInput- *processInput(null, "test")-* Execution #1	Nov. 28th, 2022	PASS
testProcessInputInvalidInput- *processInput(null, "test")-* Execution #2	Nov. 28th, 2022	PASS
testProcessInputInvalidInput- *processInput(null, "test")-* Execution #3	Nov. 28th, 2022	PASS
testProcessInputInvalidInput- *processInput(null, "test")-* Execution #4	Nov. 28th, 2022	PASS
testProcessInputInvalidInput- *processInput(null, "test")-* Execution #5	Nov. 28th, 2022	PASS
testProcessInputInvalidInput- *processInput(null, "test")-* Execution #6	Nov. 28th, 2022	PASS

Test Case #2:	**Date of Test**	PASS or

testProcessInputSourceCodeExtensionSuccessful **15/15 PASSED Execution Trials**	**Execution**	FAIL
testProcessInputSourceCodeExtensionSuccessful- *test.py* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.py* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.py* - Execution #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.java* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.java* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.java* - Execution #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.cpp* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.cpp* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.cpp* - Execution #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.c* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.c* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.c* - Execution #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.cc* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.cc* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionSuccessful- *test.cc* - Execution #3	Nov. 28th, 2022	PASS

Test Case #3: testProcessInputSourceCodeExtensionUnsuccessful **12/12 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputSourceCodeExtensionUnsuccessful-*test.txt* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.txt* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.txt* - Exc. #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.png* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.png* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.png* - Exc. #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.mp4* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.mp4* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.mp4* - Exc. #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.ml* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.ml* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeExtensionUnsuccessful-*test.ml* - Exc. #3	Nov. 28th, 2022	PASS

Test Case #4: testProcessInputSourceCodeInputUnsuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL

testProcessInputSourceCodeInputUnsuccessful- *sourceCode.setText(null)-* Execution #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeInputUnsuccessful- *sourceCode.setText(null)-* Execution #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeInputUnsuccessful- *sourceCode.setText(null)-* Execution #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeInputUnsuccessful- *sourceCode.setText(null)-* Execution #4	Nov. 28th, 2022	PASS
testProcessInputSourceCodeInputUnsuccessful- *sourceCode.setText(null)-* Execution #5	Nov. 28th, 2022	PASS
testProcessInputSourceCodeInputUnsuccessful- *sourceCode.setText(null)-* Execution #6	Nov. 28th, 2022	PASS

Test Case #5: testProcessInputLmsTargetInputSuccessful **15/15 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputLmsTargetInputSuccessful- *Canvas* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Canvas* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Canvas* - Execution #3	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *D2L* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *D2L* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *D2L* - Execution #3	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Absorb* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Absorb* - Execution #2	Nov. 28th, 2022	PASS

testProcessInputLmsTargetInputSuccessful- *Absorb* - Execution #3	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Matrix* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Matrix* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Matrix* - Execution #3	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Talent* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Talent* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputSuccessful- *Talent* - Execution #3	Nov. 28th, 2022	PASS

Test Case #6: testProcessInputLmsTargetInputUnsuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputLmsTargetInputUnsuccessful- *Expand*- Execution #1	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputUnsuccessful- *Expand*- Execution #2	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputUnsuccessful- *Expand*- Execution #3	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputUnsuccessful- *Expand*- Execution #4	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputUnsuccessful- *Expand*- Execution #5	Nov. 28th, 2022	PASS
testProcessInputLmsTargetInputUnsuccessful- *Expand*- Execution #6	Nov. 28th, 2022	PASS

Test Case #7: testProcessInputProblemTypeInputSuccessful **3/3 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputProblemTypeInputSuccessful - *Ordering* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputProblemTypeInputSuccessful - *Ordering* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputProblemTypeInputSuccessful - *Ordering* - Execution #3	Nov. 28th, 2022	PASS

Test Case #8: testProcessInputProblemTypeInputUnsuccessful **3/3 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputProblemTypeInputUnsuccessful - *Expand* - Execution #1	Nov. 28th, 2022	PASS
testProcessInputProblemTypeInputUnsuccessful - *Expand* - Execution #2	Nov. 28th, 2022	PASS
testProcessInputProblemTypeInputUnsuccessful - *Expand* - Execution #3	Nov. 28th, 2022	PASS

Test Case #9: testProcessInputProblemTtileInputSuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputProblemTtileInputSuccessful- *“test title”* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputProblemTtileInputSuccessful- *“test title”* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputProblemTtileInputSuccessful- *“test title”* - Exc. #3	Nov. 28th, 2022	PASS
testProcessInputProblemTtileInputSuccessful- *Empty String* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputProblemTtileInputSuccessful- *Empty*	Nov. 28th, 2022	PASS

String - Exc. #2		
testProcessInputProblemTtileInputSuccessful- *Empty String* - Exc. #3	Nov. 28th, 2022	PASS

<u>Test Case #10:</u> testProcessInputProblemDescriptionInputSuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputProblemDescriptionInputSuccessful- *"test description: This is a PPALMS Problem"* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputProblemDescriptionInputSuccessful- *"test description: This is a PPALMS Problem"* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputProblemDescriptionInputSuccessful- *"test description: This is a PPALMS Problem"* - Exc. #3	Nov. 28th, 2022	PASS
testProcessInputProblemDescriptionInputSuccessful- *Empty String* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputProblemDescriptionInputSuccessful- *Empty String* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputProblemDescriptionInputSuccessful- *Empty String* - Exc. #3	Nov. 28th, 2022	PASS

<u>Test Case #11:</u> testProcessInputExportProblemInputSuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputExportProblemInputSuccessful- *"exportProblem"* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputExportProblemInputSuccessful- *"exportProblem"* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputExportProblemInputSuccessful- *"exportProblem"* - Exc. #3	Nov. 28th, 2022	PASS
testProcessInputExportProblemInputSuccessful- *"exportProblem"* - Exc. #4	Nov. 28th, 2022	PASS

testProcessInputExportProblemInputSuccessful- *“exportProblem”* - Exc. #5	Nov. 28th, 2022	PASS
testProcessInputExportProblemInputSuccessful- *“exportProblem”* - Exc. #6	Nov. 28th, 2022	PASS

Test Case #12: testProcessInputSourceCodeLinesInputSuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputSourceCodeLinesInputSuccessful- *3 lines of source code* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputSuccessful- *3 lines of source code* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputSuccessful- *3 lines of source code* - Exc. #3	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputSuccessful- *3 lines of source code* - Exc. #4	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputSuccessful- *3 lines of source code* - Exc. #5	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputSuccessful- *3 lines of source code* - Exc. #6	Nov. 28th, 2022	PASS

Test Case #13: testProcessInputSourceCodeLinesInputUnsuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputSourceCodeLinesInputUnsuccessful- *0 lines of source code* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputUnsuccessful- *0 lines of source code* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputUnsuccessful- *0 lines of source code* - Exc. #3	Nov. 28th, 2022	PASS

testProcessInputSourceCodeLinesInputUnsuccessful- *51 lines of source code* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputUnsuccessful - *51 lines of source code* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputSourceCodeLinesInputUnsuccessful- *51 lines of source code* - Exc. #3	Nov. 28th, 2022	PASS

Test Case #14: testProcessInputAddAnnotationSuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputAddAnnotationSuccessful- *“addAnnotation”* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationSuccessful- *“addAnnotation”* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationSuccessful- *“addAnnotation”* - Exc. #3	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationSuccessful- *“addAnnotation”* - Exc. #4	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationSuccessful- *“addAnnotation”* - Exc. #5	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationSuccessful- *“addAnnotation”* - Exc. #6	Nov. 28th, 2022	PASS

Test Case #15: testProcessInputAddAnnotationUnsuccessful **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testProcessInputAddAnnotationUnsuccessful- *“test”* - Exc. #1	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationUnsuccessful- *“test”* - Exc. #2	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationUnsuccessful- *“test”* - Exc.	Nov. 28th, 2022	PASS

#3		
testProcessInputAddAnnotationUnsuccessful- *“test”* - Exc. #4	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationUnsuccessful- *“test”* - Exc. #5	Nov. 28th, 2022	PASS
testProcessInputAddAnnotationUnsuccessful- *“test”* - Exc. #6	Nov. 28th, 2022	PASS

3.4 PpalmsLogicHandler Test Execution Report

Test Case #1: testValidateCodeInputSuccess **15/15 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testValidateCodeInputSuccess - *test.py* - Execution #1	Nov. 25th, 2022	PASS
testValidateCodeInputSuccess - *test.py* - Execution #2	Nov. 26th, 2022	PASS
testValidateCodeInputSuccess - *test.py* - Execution #3	Nov. 27th, 2022	PASS
testValidateCodeInputSuccess - *test.java* - Execution #1	Nov. 25th, 2022	PASS
testValidateCodeInputSuccess - *test.java* - Execution #2	Nov. 26th, 2022	PASS
testValidateCodeInputSuccess - *test.java* - Execution #3	Nov. 27th, 2022	PASS
testValidateCodeInputSuccess - *test.cpp* - Execution #1	Nov. 25th, 2022	PASS
testValidateCodeInputSuccess - *test.cpp* - Execution #2	Nov. 26th, 2022	PASS
testValidateCodeInputSuccess - *test.cpp* - Execution #3	Nov. 27th, 2022	PASS
testValidateCodeInputSuccess - *test.c* - Execution #1	Nov. 25th, 2022	PASS
testValidateCodeInputSuccess - *test.c* - Execution #2	Nov. 26th, 2022	PASS
testValidateCodeInputSuccess - *test.c* - Execution #3	Nov. 27th, 2022	PASS
testValidateCodeInputSuccess - *test.cc* - Execution #1	Nov. 25th, 2022	PASS
testValidateCodeInputSuccess - *test.cc* - Execution #2	Nov. 26th, 2022	PASS

testValidateCodeInputSuccess - *test.cc* - Execution #3	Nov. 27th, 2022	PASS

Test Case #2: testValidateCodeInputNoCode **3/3 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testValidateCodeInputNoCode - Execution #1	Nov. 25th, 2022	PASS
testValidateCodeInputNoCode - Execution #2	Nov. 26th, 2022	PASS
testValidateCodeInputNoCode - Execution #3	Nov. 27th, 2022	PASS

Test Case #3: testValidateCodeInputInvalidExtension **12/12 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testValidateCodeInputInvalidExtension - *test.txt* - Exc. #1	Nov. 25th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.txt* - Exc. #2	Nov. 26th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.txt* - Exc. #3	Nov. 27th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.png* - Exc. #1	Nov. 25th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.png* - Exc. #2	Nov. 26th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.png* - Exc. #3	Nov. 27th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.mp4* - Exc. #1	Nov. 25th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.mp4* - Exc. #2	Nov. 26th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.mp4* - Exc. #3	Nov. 27th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.ml* - Exc. #1	Nov. 25th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.ml* - Exc. #2	Nov. 26th, 2022	PASS
testValidateCodeInputInvalidExtension - *test.ml* - Exc. #3	Nov. 27th, 2022	PASS

Test Case #4: testValidateTitleInputSuccess **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testValidateTitleInputSuccess - *“Test”* - Exc. #1	Nov. 25th, 2022	PASS
testValidateTitleInputSuccess - *“Test”* - Exc. #2	Nov. 26th, 2022	PASS
testValidateTitleInputSuccess - *“Test”* - Exc. #3	Nov. 27th, 2022	PASS

testValidateTitleInputSuccess - *Empty String* - Exc. #1	Nov. 25th, 2022	PASS
testValidateTitleInputSuccess - *Empty String* - Exc. #2	Nov. 26th, 2022	PASS
testValidateTitleInputSuccess - *Empty String* - Exc. #3	Nov. 27th, 2022	PASS

Test Case #5: testValidateTitleInputFailure **3/3 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testValidateTitleInputFailure - *null* - Exc. #1	Nov. 25th, 2022	PASS
testValidateTitleInputFailure - *null* - Exc. #2	Nov. 26th, 2022	PASS
testValidateTitleInputFailure - *null* - Exc. #3	Nov. 27th, 2022	PASS

Test Case #6: testValidateDescInputSuccess **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testValidateDescInputSuccess - *"Test"* - Exc. #1	Nov. 25th, 2022	PASS
testValidateDescInputSuccess - *"Test"* - Exc. #2	Nov. 26th, 2022	PASS
testValidateDescInputSuccess - *"Test"* - Exc. #3	Nov. 27th, 2022	PASS
testValidateDescInputSuccess - *Empty String* - Exc. #1	Nov. 25th, 2022	PASS
testValidateDescInputSuccess - *Empty String* - Exc. #2	Nov. 26th, 2022	PASS
testValidateDescInputSuccess - *Empty String* - Exc. #3	Nov. 27th, 2022	PASS

Test Case #7: testValidateDescInputFailure **3/3 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testValidateDescInputFailure - *null* - Exc. #1	Nov. 25th, 2022	PASS
testValidateDescInputFailure - *null* - Exc. #2	Nov. 26th, 2022	PASS
testValidateDescInputFailure - *null* - Exc. #3	Nov. 27th, 2022	PASS

Test Case #8: testSetAnnotations **9/9 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testSetAnnotations - *In-Order* - Execution #1	Nov. 25th, 2022	PASS

testSetAnnotations - *In-Order* - Execution #2	Nov. 26th, 2022	PASS
testSetAnnotations - *In-Order* - Execution #3	Nov. 27th, 2022	PASS
testSetAnnotations - *Not In-Order* - Execution #1	Nov. 25th, 2022	PASS
testSetAnnotations - *Not In-Order* - Execution #2	Nov. 26th, 2022	PASS
testSetAnnotations - *Not In-Order* - Execution #3	Nov. 27th, 2022	PASS
testSetAnnotations - *Minimal Selection* - Execution #1	Nov. 25th, 2022	PASS
testSetAnnotations - *Minimal Selection* - Execution #2	Nov. 26th, 2022	PASS
testSetAnnotations - *Minimal Selection* - Execution #3	Nov. 27th, 2022	PASS

Test Case #9: testCreatePermutations **12/12 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testCreatePermutations - *One-Line* - Execution #1	Nov. 25th, 2022	PASS
testCreatePermutations - *One-Line* - Execution #2	Nov. 26th, 2022	PASS
testCreatePermutations - *One-Line* - Execution #3	Nov. 27th, 2022	PASS
testCreatePermutations - *Multiple-Lines-Below-Limit* - Execution #1	Nov. 25th, 2022	PASS
testCreatePermutations - *Multiple-Lines-Below-Limit* - Execution #2	Nov. 26th, 2022	PASS
testCreatePermutations - *Multiple-Lines-Below-Limit* - Execution #3	Nov. 27th, 2022	PASS
testCreatePermutations - *Multiple-Lines-Above-Limit* - Execution #1	Nov. 25th, 2022	PASS
testCreatePermutations - *Multiple-Lines-Above-Limit* - Execution #2	Nov. 26th, 2022	PASS
testCreatePermutations - *Multiple-Lines-Above-Limit* - Execution #3	Nov. 27th, 2022	PASS
testCreatePermutations - *Fifty-Lines* - Execution #1	Nov. 25th, 2022	PASS
testCreatePermutations - *Fifty-Lines* - Execution #2	Nov. 26th, 2022	PASS

testCreatePermutations - *Fifty-Lines* - Execution #3	Nov. 27th, 2022	PASS

Test Case #10: testExportPpalmsProblem **6/6 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testExportPpalmsProblem - *Title-Descripton-Canvas-Target-LMS* - Execution #1	Nov. 25th, 2022	PASS
testExportPpalmsProblem - *Title-Descripton-Canvas-Target-LMS* - Execution #2	Nov. 26th, 2022	PASS
testExportPpalmsProblem - *Title-Descripton-Canvas-Target-LMS* - Execution #3	Nov. 27th, 2022	PASS
testExportPpalmsProblem - *No-Title-No-Descripton-D2L-Target-LMS* - Execution #1	Nov. 25th, 2022	PASS
testExportPpalmsProblem - *No-Title-No-Descripton-D2L-Target-LMS* - Execution #2	Nov. 26th, 2022	PASS
testExportPpalmsProblem - *No-Title-No-Descripton-D2L-Target-LMS* - Execution #3	Nov. 27th, 2022	PASS

Test Case #11: testExportPpalmsProblemFail **3/3 PASSED Execution Trials**	**Date of Test Execution**	PASS or FAIL
testExportPpalmsProblemFail - *File-Conflict* - Execution #1	Nov. 25th, 2022	PASS
testExportPpalmsProblemFail - *File-Conflict* - Execution #2	Nov. 26th, 2022	PASS
testExportPpalmsProblemFail - *File-Conflict* - Execution #3	Nov. 27th, 2022	PASS

4. End-to-End Test Scenario

This section contains a detailed, end-to-end test scenario that tests the entirety of the application. This test scenario will explore each module of the application and contains the general flow of using the application. Following the following introductory paragraphs, the end-to-end test scenario is divided into sequential steps the user will take when performing the scenario described in the section paragraph. Due to the nature of the PPALMS application utilizing a graphical user interface, this scenario also provides screenshots of what will be seen during this scenario. It is important to note that the screenshots provided were run on an Apple MacBook Pro utilizing the Mac operating system. The application will run the same and should expect the same performance as long as the user follows the hardware expectations listed in the SRS document.

Specifically, the end-to-end test scenario will be a general use of the application. This means that all of the use cases of the application will be highlighted in this end-to-end test scenario. The user will be able to open the PPALMS application (SRS 2.1); input their source code file for problem creation (SRS 5.1); select their LMS target (SRS 5.2), select problem type (SRS 5.3) (As stated in the design document, the only problem type is ordering, therefore this scenario will incorporate the ordering problem type); select annotations (SRS 5.4) (PPALMS version 0.0.1 only allows for ordering problems, this means the annotations will represent the lines of code the user wishes to include in their problem as mentioned in the SRS document), provide an optional title/description (SRS 5.5); and export their problem (SRS 5.6).

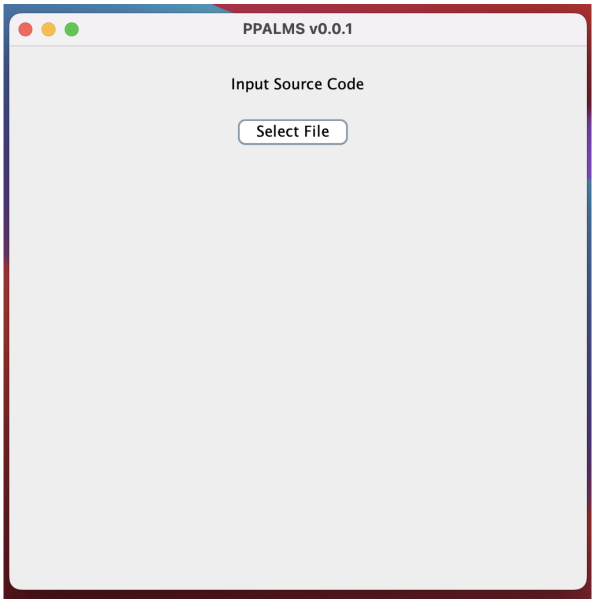
At the end of this process, the user will have a JSON file that corresponds to the problem that they created using the PPALMS application (SRS 5.7). The use of a JSON beautifier will be used to display the JSON in a readable format for the purpose of this scenario. However, it is important to note that using a JSON beautifier does not change the contents of the JSON file, it only changes how it is visually represented.

Setup: The setup required for the PPALMS application is found in the README file that has been distributed with the release of the application. Inside of the README the user will note that they must have a Java Runtime Environment to run the application as also mentioned in the design document (Section 1.4 Design Objectives) and requirements specification document (SRS 2.2 Hardware Requirements). Upon successful validation of an installed Java Runtime Environment. The user has been assumed to give permissions to run Java applications. This process is also described in the README file. After this setup, the user can open the application as an executable JAR file. (They can run the program.)

Step 1: Execute (Run) the PPALMS application.

- The following screenshot displays the PPALMS application after it has been launched.

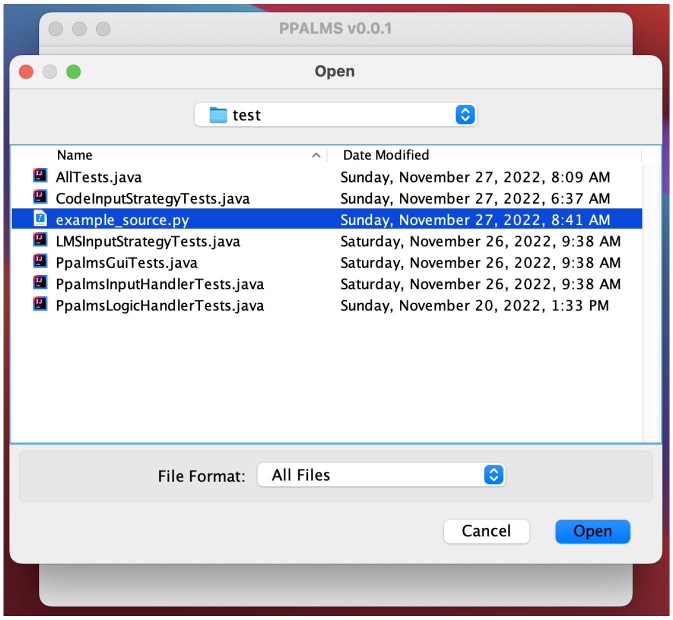
- We can verify that the PPALMS application is running when we see the following window appear on our screen. This verifies that the PPALMS application is running successfully.



The PPALMS application after executing the JAR.

Step 2: This step follows SRS 5.1 for inputting source code.
Click the "Select File" button to input a source code file to the application. Navigate to the source code file to select, then press "Open" on the file chooser dialog menu.

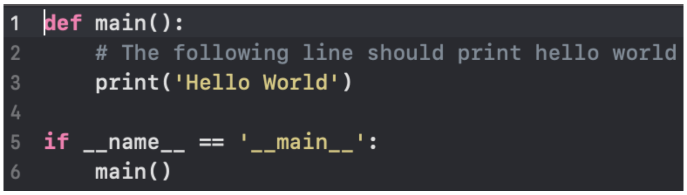
- The following screenshots include the PPALMS application inside of the file chooser dialog menu with the highlighted selected file "example_source.py"
- The contents of the file "example_source.py" are shown in the second screenshot to give the reader context to what is included in the test file.
- We can verify the chosen SourceCode file is valid, of reasonable length (1-50 lines) as the view of the PPALMS application has changed and no error dialog was presented.



Name	Date Modified
AllTests.java	Sunday, November 27, 2022, 8:09 AM
CodeInputStrategyTests.java	Sunday, November 27, 2022, 6:37 AM
example_source.py	Sunday, November 27, 2022, 8:41 AM
LMSInputStrategyTests.java	Saturday, November 26, 2022, 9:38 AM
PpalmsGuiTests.java	Saturday, November 26, 2022, 9:38 AM
PpalmsInputHandlerTests.java	Saturday, November 26, 2022, 9:38 AM
PpalmsLogicHandlerTests.java	Sunday, November 20, 2022, 1:33 PM

File Format: All Files

The file chooser menu dialog.



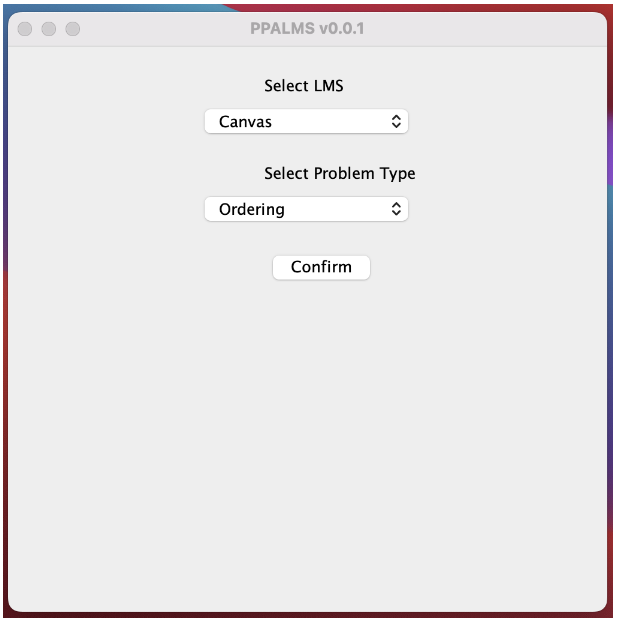
```python
def main():
    # The following line should print hello world
    print('Hello World')

if __name__ == '__main__':
    main()
```

The contents of “example_source.py” used in the end-to-end test scenario.

Step 3: This step follows SRS 5.2 for selecting a target LMS and SRS 5.3 for selecting a Parson’s problem type.

Select a target LMS from the combo box associated with the “Select LMS” label. Then, select the problem type from the combo box associated with the “Select Problem Type” label. The confirm button becomes available to be selected after selecting an input for both of the respective fields. This button will be clicked upon availability.

- In this test scenario, the target LMS that is selected is “Canvas” and the problem type selected is “Ordering”.
- We can verify a target LMS and Problem type is selected when the confirm button is enabled. Clicking the confirm button would result in a change of view, an indication of a successful process. Due to the nature of the graphical user interface’s ComboBoxes, there are only a set amount of fields that the user can enter. Therefore, we can verify that the selections of the LMS and problem type are valid.



Select LMS

Canvas

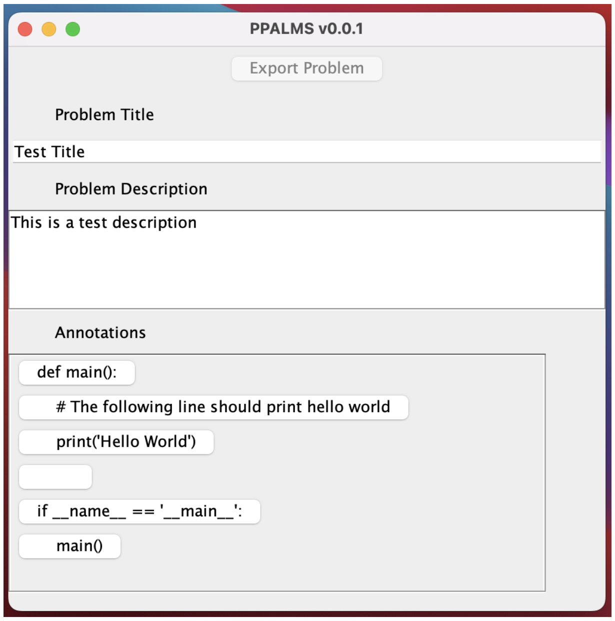
Select Problem Type

Ordering

The PPALMS application interface for selecting LMS target and problem type.

Step 4: This step follows SRS 5.5 for providing an optional title and description. Enter a title and description for the problem by utilizing the respective text fields corresponding to the labels.

- In this test scenario, the title “Test Title” is given.
- In this test scenario, the description “This is a test description” is given.
- As both the problem title and description are optional, verification is not required for this. However, due to the nature of the graphical user interface, any input in these text fields would be sufficient. Therefore, we can verify that the PPALMS system is handling this correctly.



Problem Title

Test Title

Problem Description

This is a test description

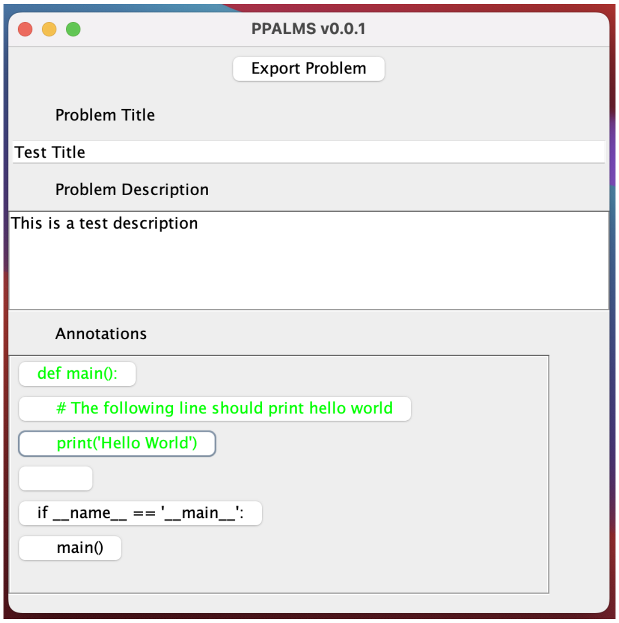
Annotations

```
def main():
    # The following line should print hello world
    print('Hello World')

if __name__ == '__main__':
    main()
```

The PPALMS problem input view with title and description specified.

Step 5: This step follows SRS 5.4 for including/excluding annotations.
Select problem annotations.

- As already mentioned, this test scenario uses the ordering problem type. The annotations are selected by clicking on the button corresponding to the line of code that the user wants for their problem. In the case of this test scenario, the user selects the main function of the source code lines.
- After a selection is made, the line will turn green indicating that it has been annotated. The following screenshot shows problem annotations for the test scenario.
- The only verification for the proposed annotations are the indications for lines in green upon clicking on them. When the line turns green, this provides enough information to verify that the problem annotation was accepted to the system.



Problem Title

Test Title

Problem Description

This is a test description

```
def main():
    # The following line should print hello world
    print('Hello World')

if __name__ == '__main__':
    main()
```

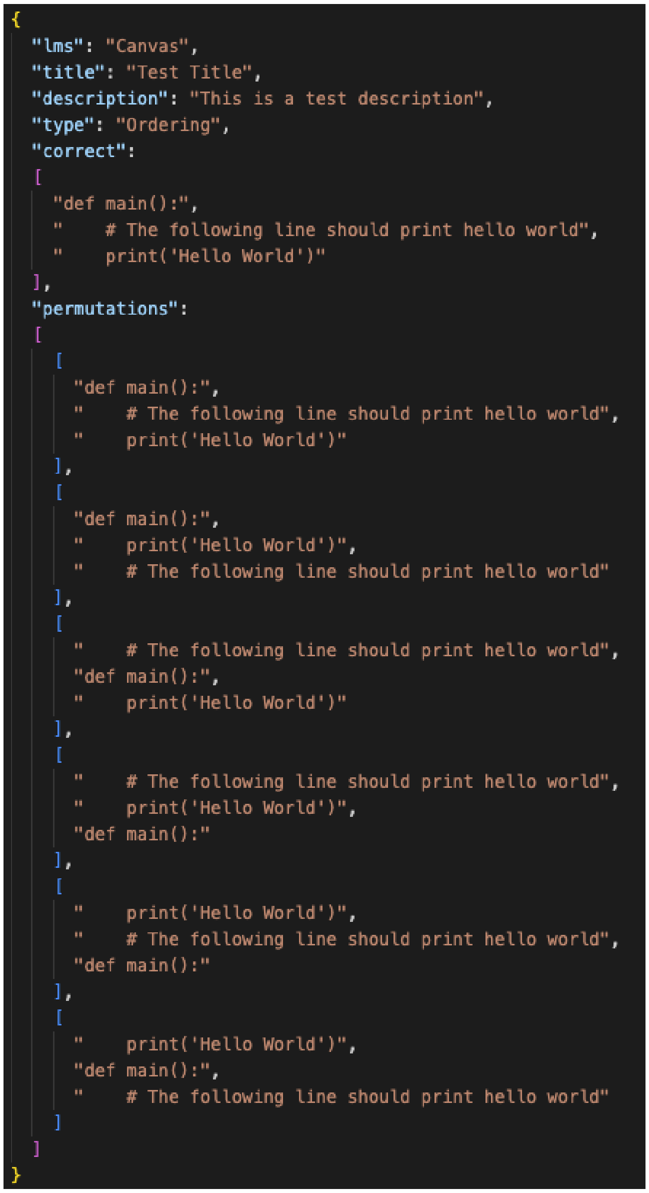
The PPALMS problem input view with selected annotations for an ordering problem.

Step 6: This step follows SRS 5.6 for validating for file output and SRS 5.7 for generating permutations in the form of a JSON file.
Export the PPALMS problem by clicking the "Export Problem" button.

- This step will generate the JSON file for our PPALMS problem as described in the design document. The JSON file will be in the same location that the application was run in. The following screenshot shows the JSON file output for the test scenario.
- As described above, the screenshot here uses a JSON beautifier to make the following screenshot readable. The PPALMS application does not handle beautifying the JSON.
- Upon this operation, the PPALMS application will close.
- We verify this for which a JSON file is generated in the same directory for which the application is executed. Given that no error dialogs are presented, the export process

would then be verified as a success since we are able to view the contents of the file and verify that it is what is expected of our system.

This marks the end of the end-to-end test scenario. This process showed the general use of the PPALMS system and the steps that will be taken while using the application. The program was able to receive inputs described in the design document and the requirements specification and able to produce the expected output JSON file.



```
{
  "lms": "Canvas",
  "title": "Test Title",
  "description": "This is a test description",
  "type": "Ordering",
  "correct":
  [
    "def main():",
    "    # The following line should print hello world",
    "    print('Hello World')"
  ],
  "permutations":
  [
    [
      "def main():",
      "    # The following line should print hello world",
      "    print('Hello World')"
    ],
    [
      "def main():",
      "    print('Hello World')",
      "    # The following line should print hello world"
    ],
    [
      "    # The following line should print hello world",
      "def main():",
      "    print('Hello World')"
    ],
    [
      "    # The following line should print hello world",
      "    print('Hello World')",
      "def main():"
    ],
    [
      "    print('Hello World')",
      "    # The following line should print hello world",
      "def main():"
    ],
    [
      "    print('Hello World')",
      "def main():",
      "    # The following line should print hello world"
    ]
  ]
}
```

A beautified version of the output JSON for the created PPALMS problem.